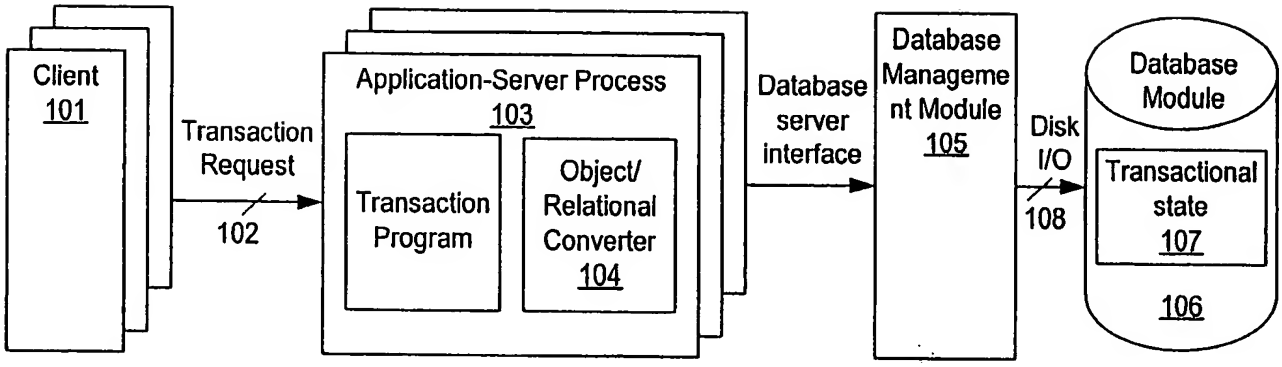

Client
101
Transaction
Request
102
Application-Server Process
103
Transaction
Program
Object/
Relational
Converter
104
Database
server
interface
Database
Manageme
nt Module
105
Disk
I/O
108
Database
Module
Transactional
state
107
106

FIG. 1

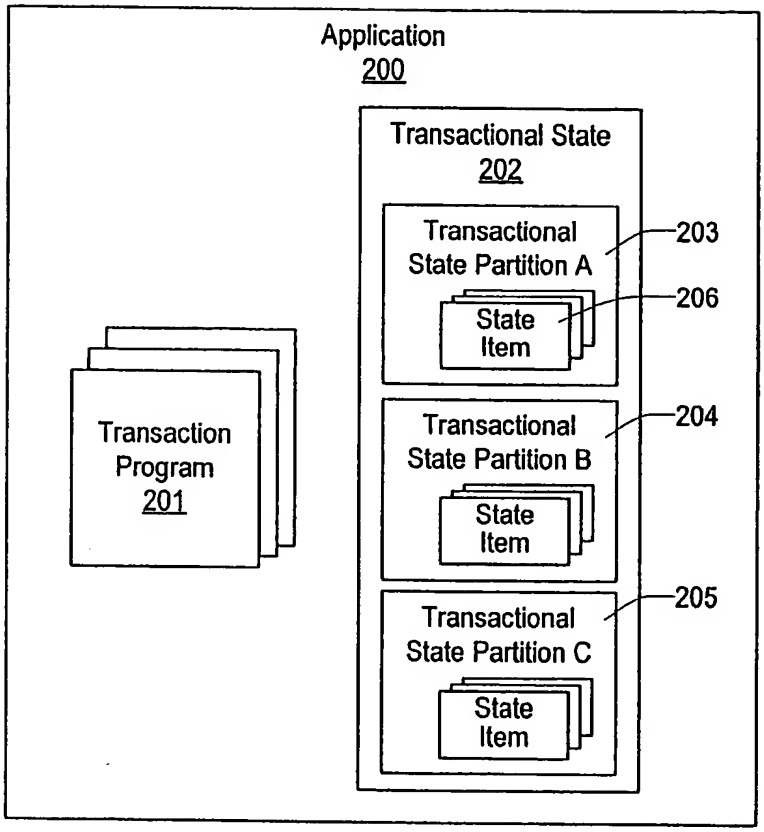

Application
200
Transactional State
202
Transactional
State Partition A
203
State
Item
206
Transactional
State Partition B
204
State
Item
Transactional
State Partition C
205
State
Item
Transaction
Program
201

FIG. 2

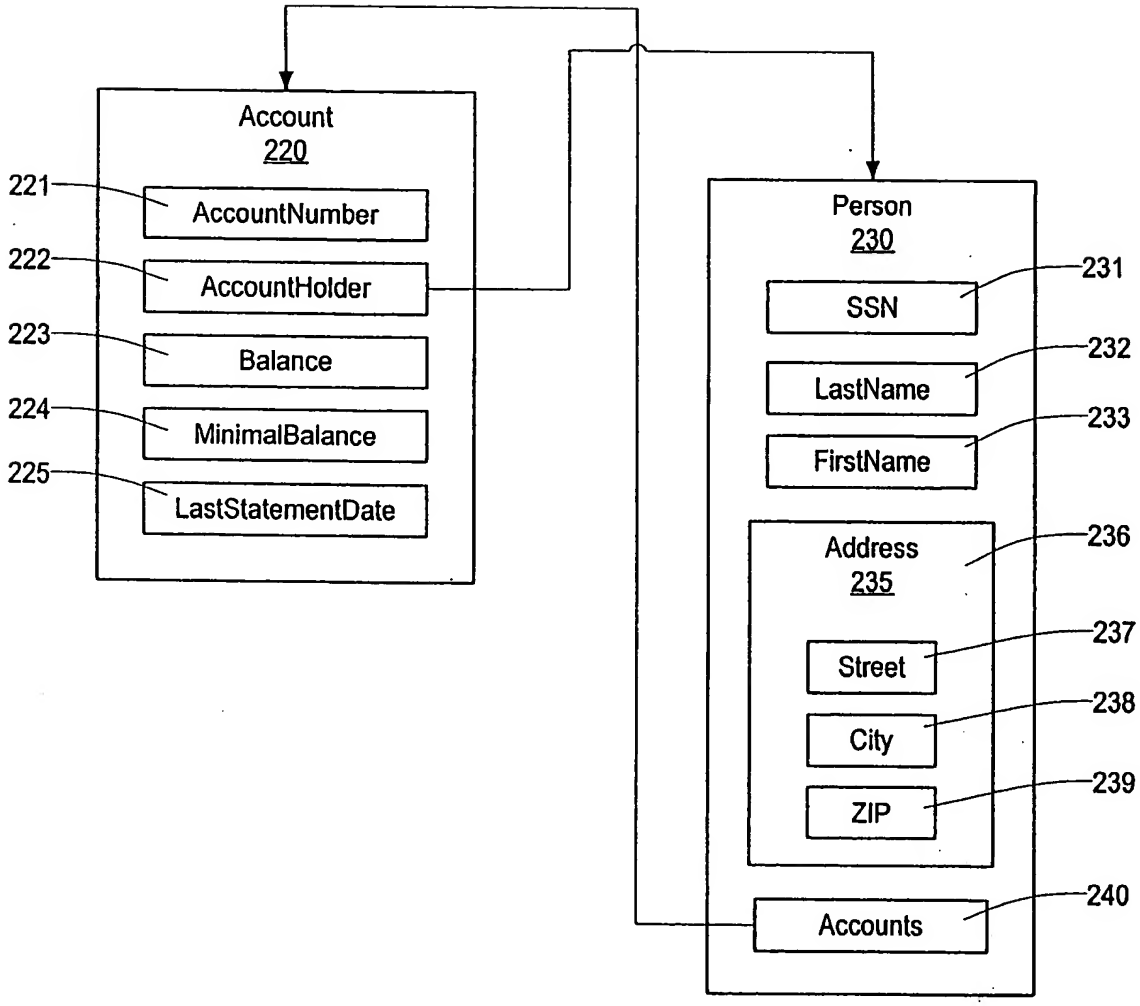

Account
220
221
AccountNumber
222
AccountHolder
223
Balance
224
MinimalBalance
225
LastStatementDate
Person
230
231
SSN
232
LastName
233
FirstName
Address
235
236
237
Street
238
City
239
ZIP
240
Accounts

FIG. 2-A

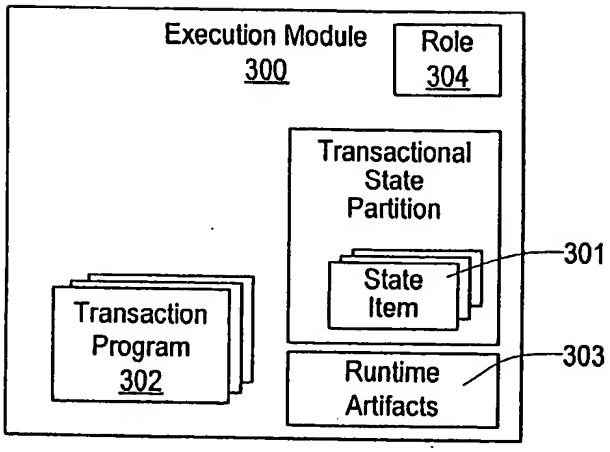

Execution Module
300
Role
304
Transactional
State
Partition
301
State
Item
Transaction
Program
302
303
Runtime
Artifacts

FIG. 3

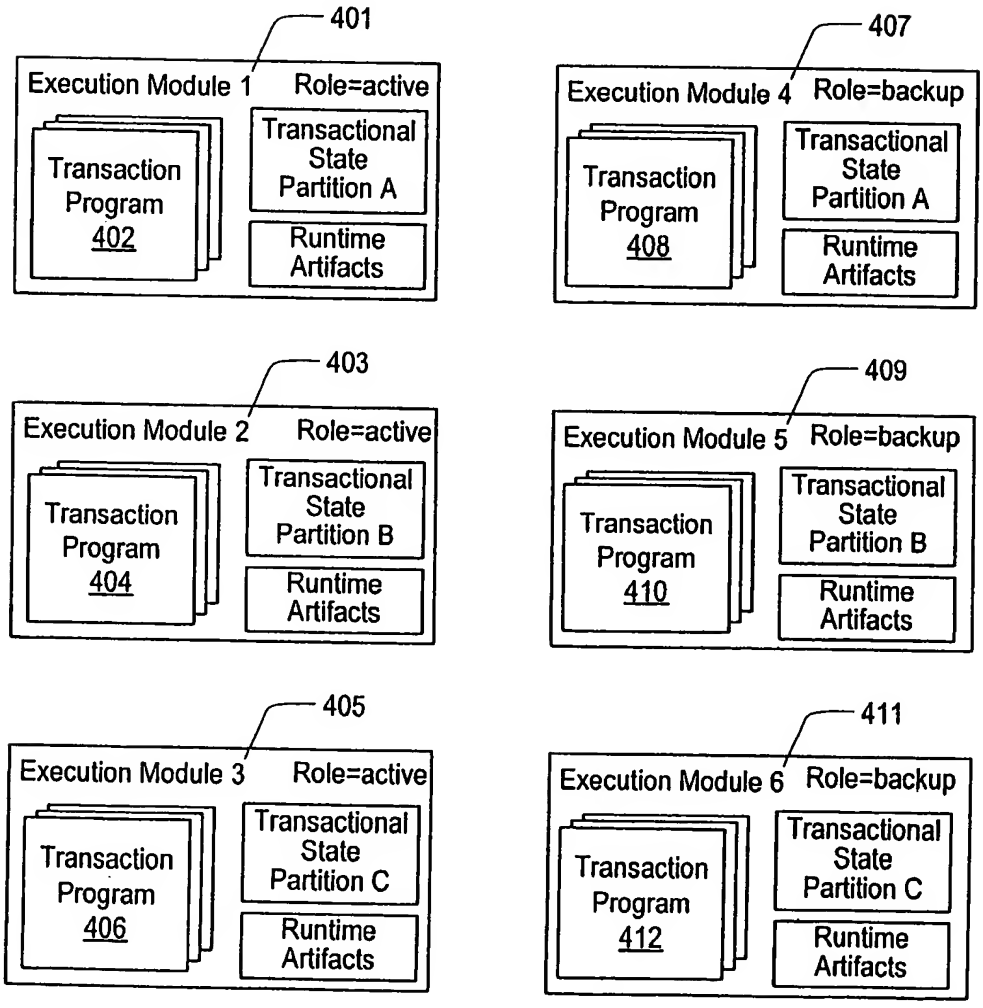

401
Execution Module 1
Role=active
Transaction
Program
402
Transactional
State
Partition A
Runtime
Artifacts
407
Execution Module 4
Role=backup
Transaction
Program
408
Transactional
State
Partition A
Runtime
Artifacts
403
Execution Module 2
Role=active
Transaction
Program
404
Transactional
State
Partition B
Runtime
Artifacts
409
Execution Module 5
Role=backup
Transaction
Program
410
Transactional
State
Partition B
Runtime
Artifacts
405
Execution Module 3
Role=active
Transaction
Program
406
Transactional
State
Partition C
Runtime
Artifacts
411
Execution Module 6
Role=backup
Transaction
Program
412
Transactional
State
Partition C
Runtime
Artifacts

FIG. 4

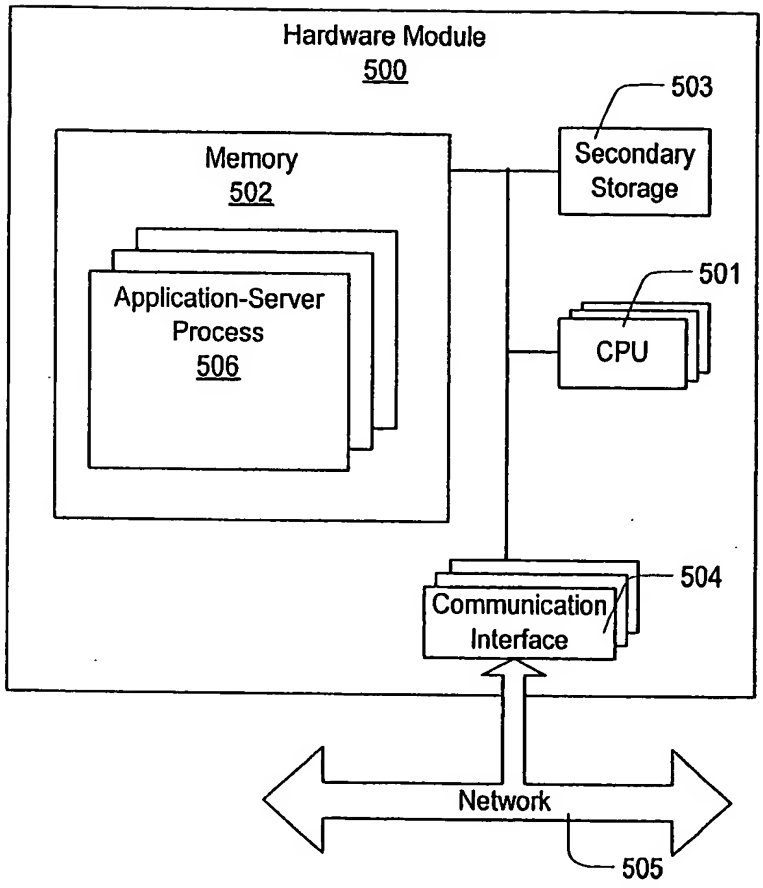

Hardware Module
500
503
Secondary Storage
Memory
502
Application-Server Process
506
501
CPU
504
Communication Interface
Network
505

FIG. 5

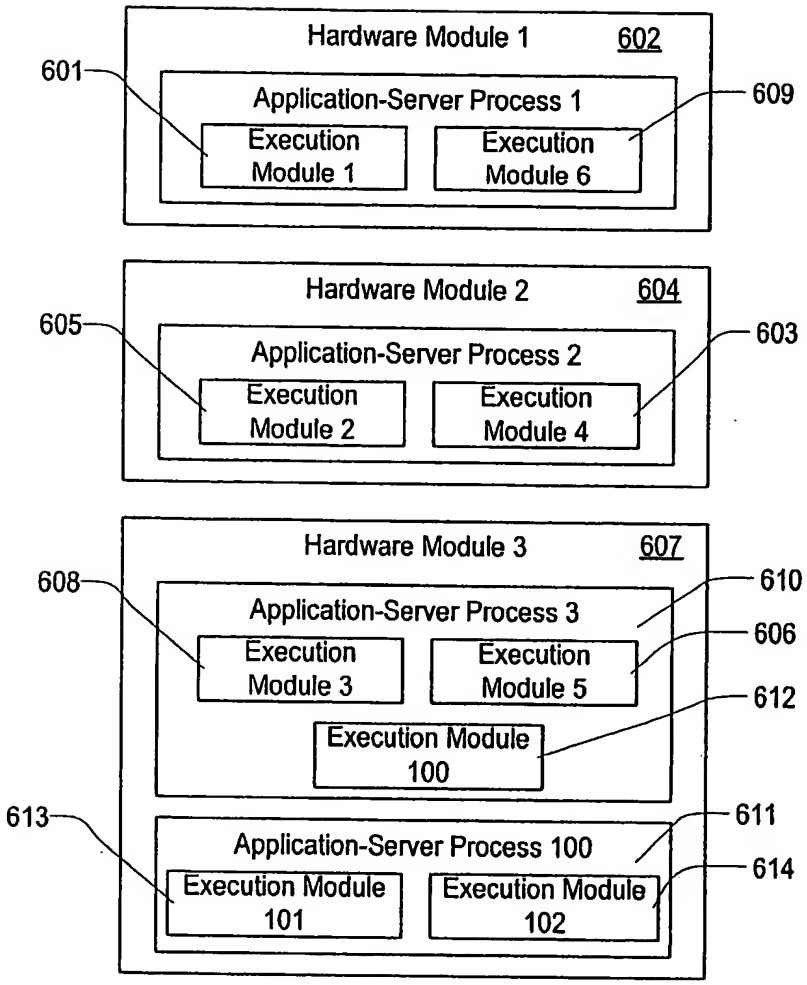

Hardware Module 1
602
601
Application-Server Process 1
609
Execution Module 1
Execution Module 6
Hardware Module 2
604
605
Application-Server Process 2
603
Execution Module 2
Execution Module 4
Hardware Module 3
607
608
610
Application-Server Process 3
606
Execution Module 3
Execution Module 5
612
Execution Module 100
613
611
Application-Server Process 100
614
Execution Module 101
Execution Module 102

FIG. 6

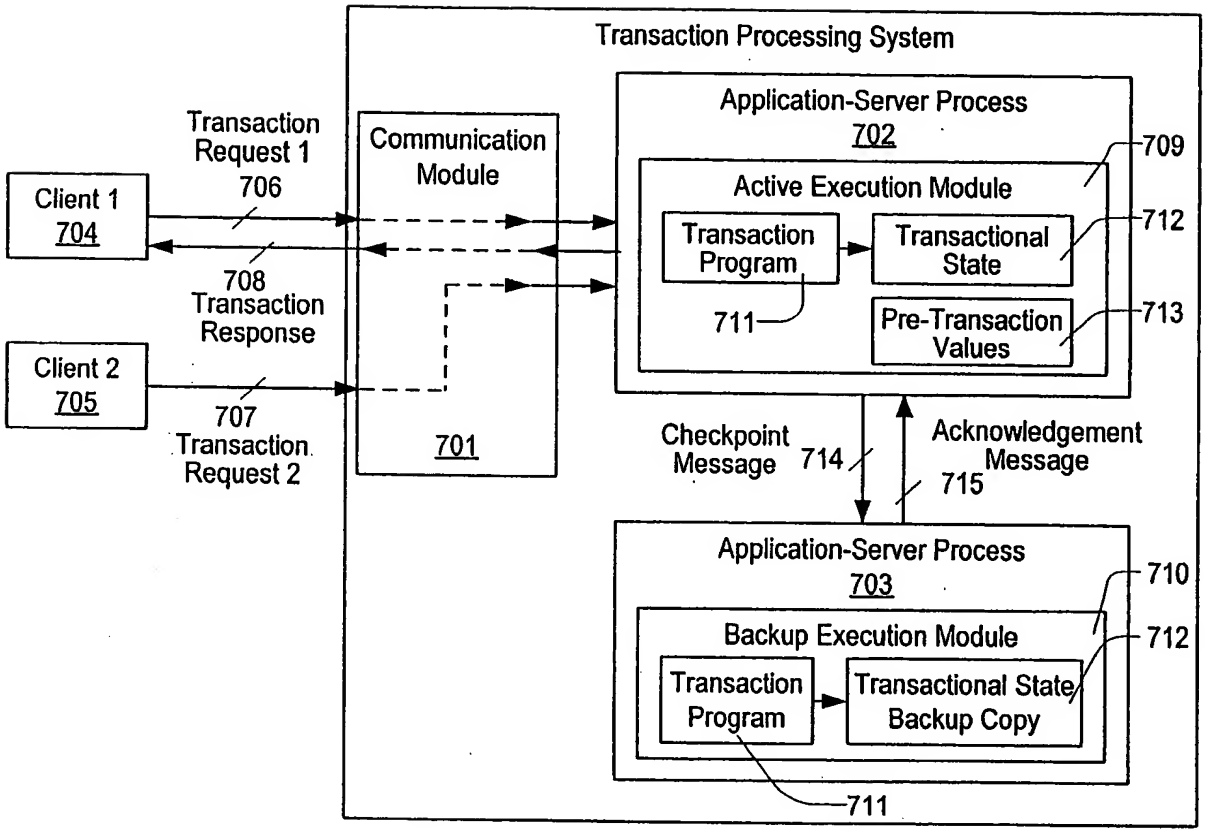

Transaction Processing System
Application-Server Process
702
Communication Module
Transaction Request 1
706
Client 1
704
708
Transaction Response
Client 2
705
707
Transaction Request 2
701
Active Execution Module
709
Transaction Program
Transactional State
712
711
Pre-Transaction Values
713
Checkpoint Message
714
Acknowledgement Message
715
Application-Server Process
703
710
Backup Execution Module
712
Transaction Program
Transactional State Backup Copy
711

FIG. 7

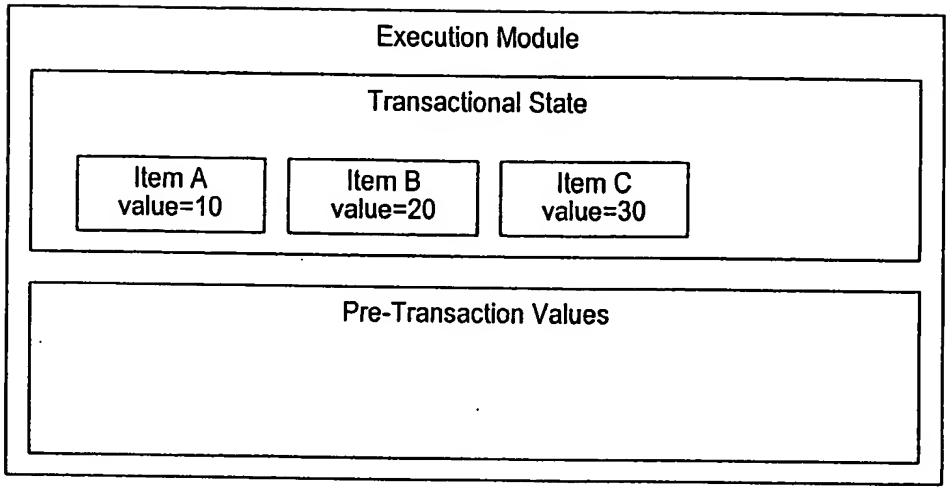

Execution Module
Transactional State
Item A
value=10
Item B
value=20
Item C
value=30
Pre-Transaction Values

FIG. 8

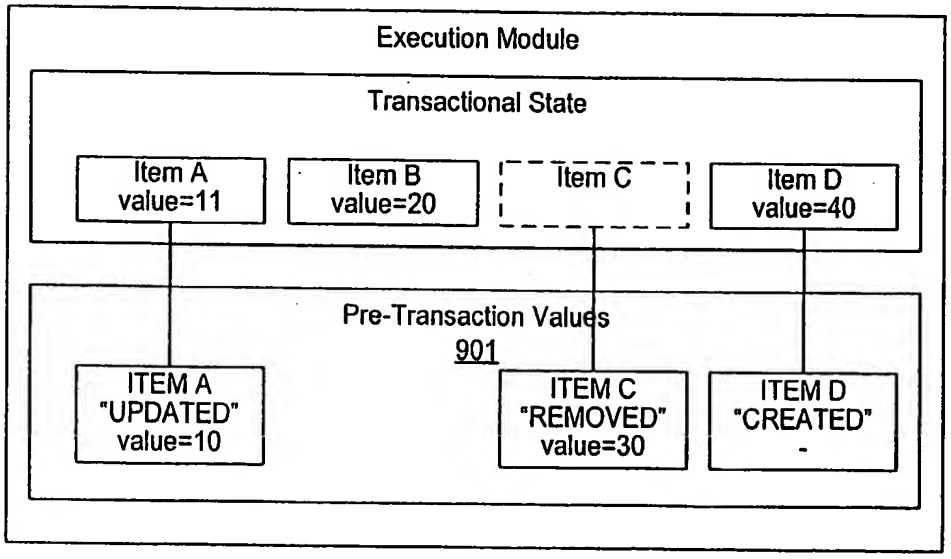

Execution Module
Transactional State
Item A
value=11
Item B
value=20
Item C
Item D
value=40
Pre-Transaction Values
901
ITEM A
"UPDATED"
value=10
ITEM C
"REMOVED"
value=30
ITEM D
"CREATED"
-

FIG. 9

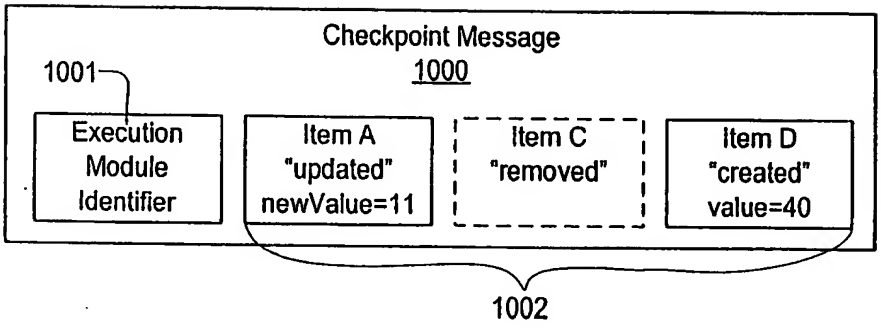

Checkpoint Message
1000
1001
Execution Module Identifier
Item A "updated" newValue=11
Item C "removed"
Item D "created" value=40
1002

FIG. 10

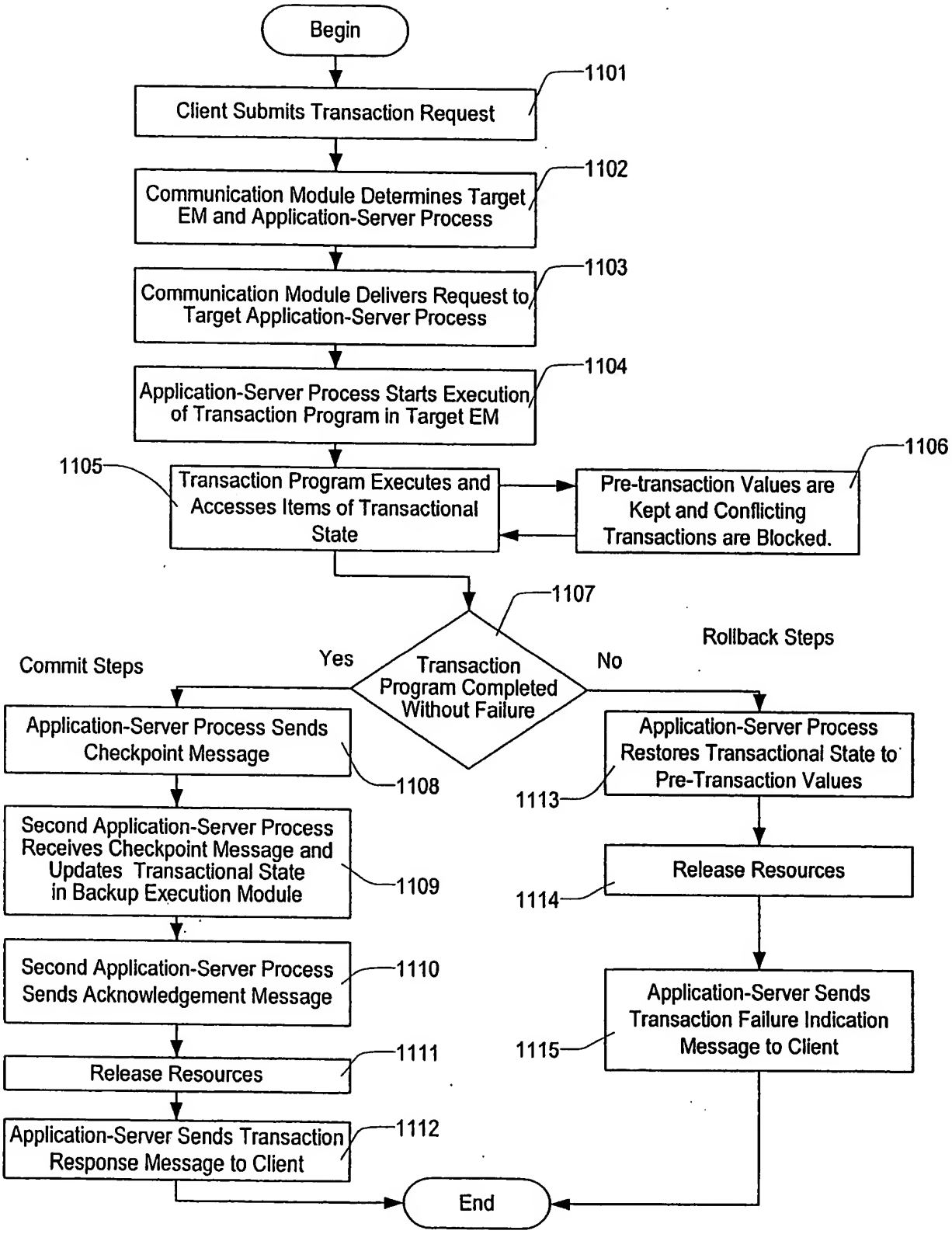

Begin
Client Submits Transaction Request
1101
Communication Module Determines Target EM and Application-Server Process
1102
Communication Module Delivers Request to Target Application-Server Process
1103
Application-Server Process Starts Execution of Transaction Program in Target EM
1104
1105
Transaction Program Executes and Accesses Items of Transactional State
Pre-transaction Values are Kept and Conflicting Transactions are Blocked.
1106
1107
Transaction Program Completed Without Failure
Yes
No
Commit Steps
Rollback Steps
Application-Server Process Sends Checkpoint Message
1108
Second Application-Server Process Receives Checkpoint Message and Updates Transactional State in Backup Execution Module
1109
Second Application-Server Process Sends Acknowledgement Message
1110
Release Resources
1111
Application-Server Sends Transaction Response Message to Client
1112
Application-Server Process Restores Transactional State to Pre-Transaction Values
1113
Release Resources
1114
Application-Server Sends Transaction Failure Indication Message to Client
1115
End

FIG. 11

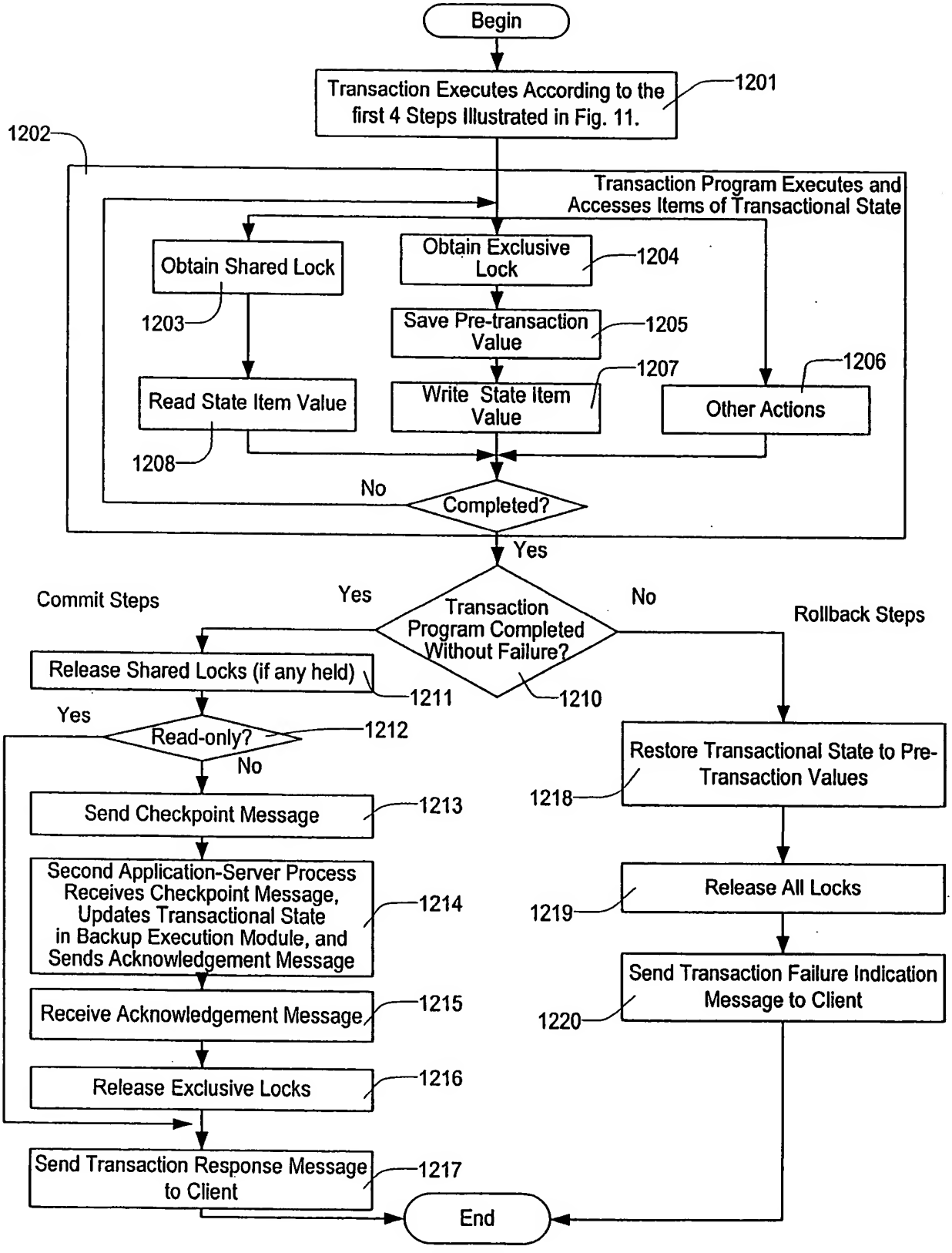

Begin
Transaction Executes According to the first 4 Steps Illustrated in Fig. 11.
1201
1202
Transaction Program Executes and Accesses Items of Transactional State
Obtain Shared Lock
1203
Obtain Exclusive Lock
1204
Save Pre-transaction Value
1205
Read State Item Value
1208
Write State Item Value
1207
Other Actions
1206
No
Completed?
Yes
Commit Steps
Yes
Transaction Program Completed Without Failure?
No
Rollback Steps
1210
Release Shared Locks (if any held)
1211
Yes
Read-only?
1212
No
Send Checkpoint Message
1213
Second Application-Server Process Receives Checkpoint Message, Updates Transactional State in Backup Execution Module, and Sends Acknowledgement Message
1214
Receive Acknowledgement Message
1215
Release Exclusive Locks
1216
Send Transaction Response Message to Client
1217
Restore Transactional State to Pre-Transaction Values
1218
Release All Locks
1219
Send Transaction Failure Indication Message to Client
1220
End

FIG. 12

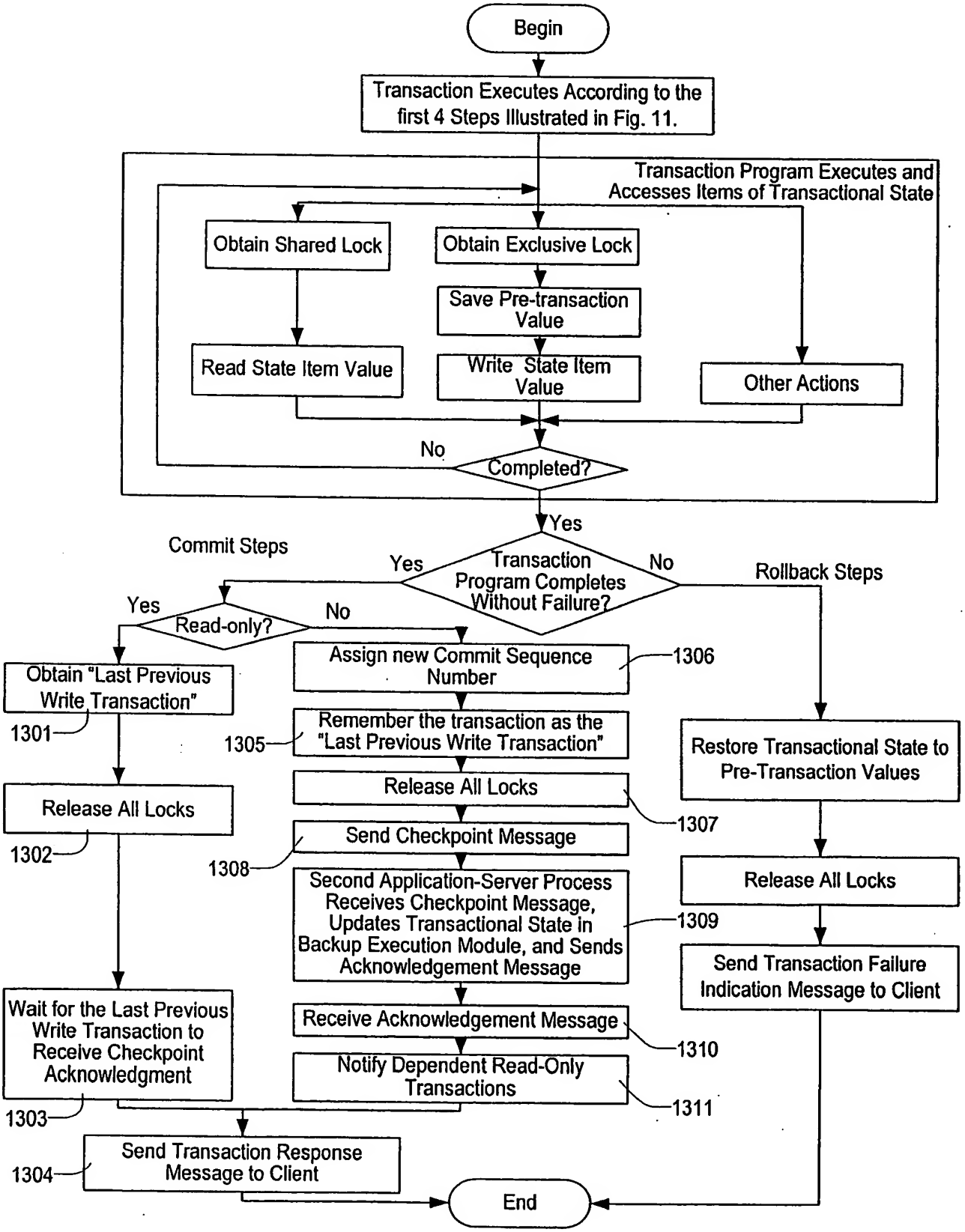

Begin
Transaction Executes According to the first 4 Steps Illustrated in Fig. 11.
Transaction Program Executes and Accesses Items of Transactional State
Obtain Shared Lock
Obtain Exclusive Lock
Save Pre-transaction Value
Read State Item Value
Write State Item Value
Other Actions
No
Completed?
Yes
Commit Steps
Yes
Transaction Program Completes Without Failure?
No
Rollback Steps
Yes
Read-only?
No
Obtain "Last Previous Write Transaction"
1301
Assign new Commit Sequence Number
1306
Remember the transaction as the "Last Previous Write Transaction"
1305
Restore Transactional State to Pre-Transaction Values
Release All Locks
1302
Release All Locks
1307
Send Checkpoint Message
1308
Release All Locks
Second Application-Server Process Receives Checkpoint Message, Updates Transactional State in Backup Execution Module, and Sends Acknowledgement Message
1309
Send Transaction Failure Indication Message to Client
Wait for the Last Previous Write Transaction to Receive Checkpoint Acknowledgment
Receive Acknowledgement Message
1310
Notify Dependent Read-Only Transactions
1311
1303
Send Transaction Response Message to Client
1304
End

FIG. 13

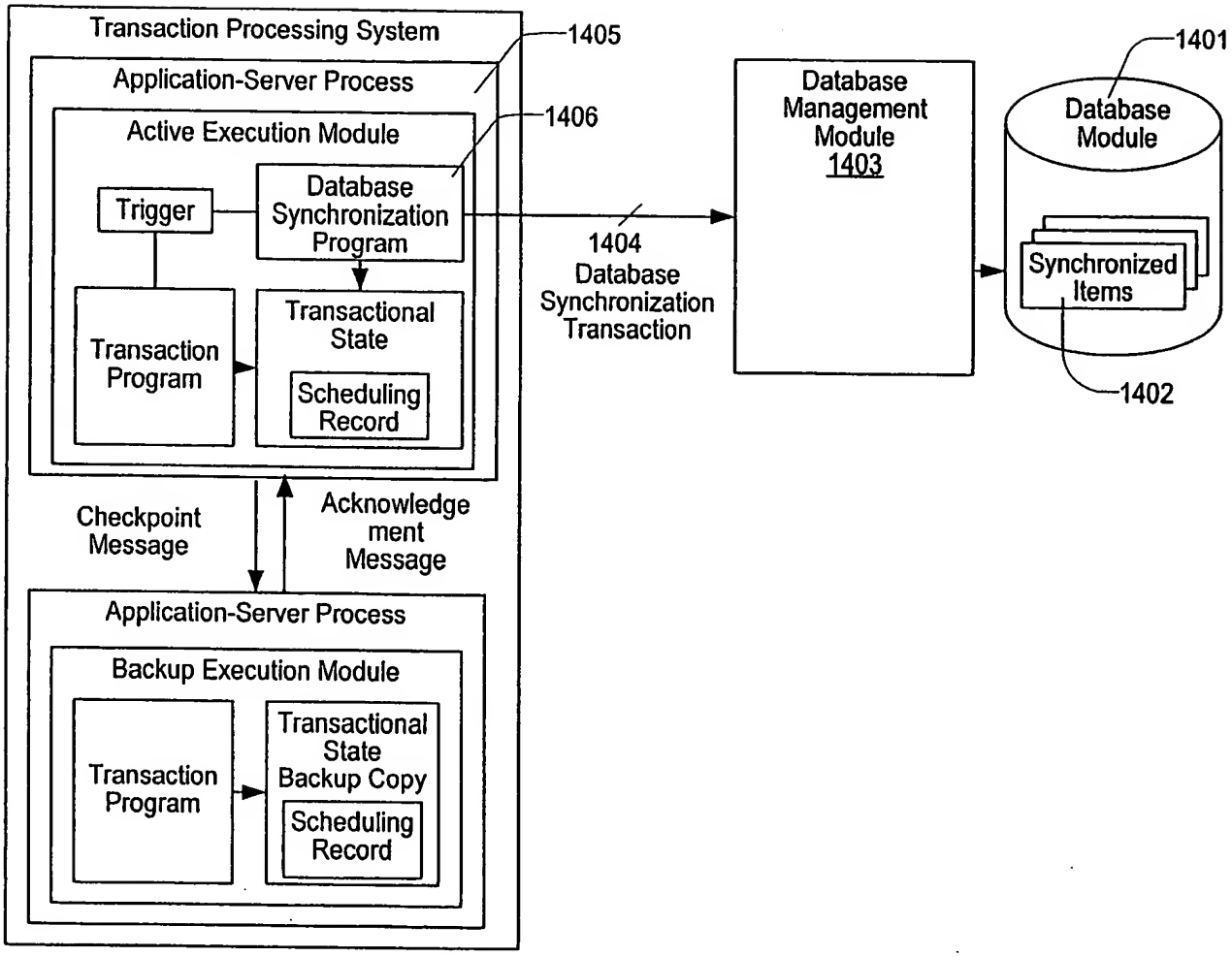

Transaction Processing System
1405
Application-Server Process
1406
Active Execution Module
Trigger
Database Synchronization Program
Transaction Program
Transactional State
Scheduling Record
1404
Database Synchronization Transaction
Database Management Module
1403
1401
Database Module
Synchronized Items
1402
Checkpoint Message
Acknowledgement Message
Application-Server Process
Backup Execution Module
Transaction Program
Transactional State Backup Copy
Scheduling Record

FIG. 14

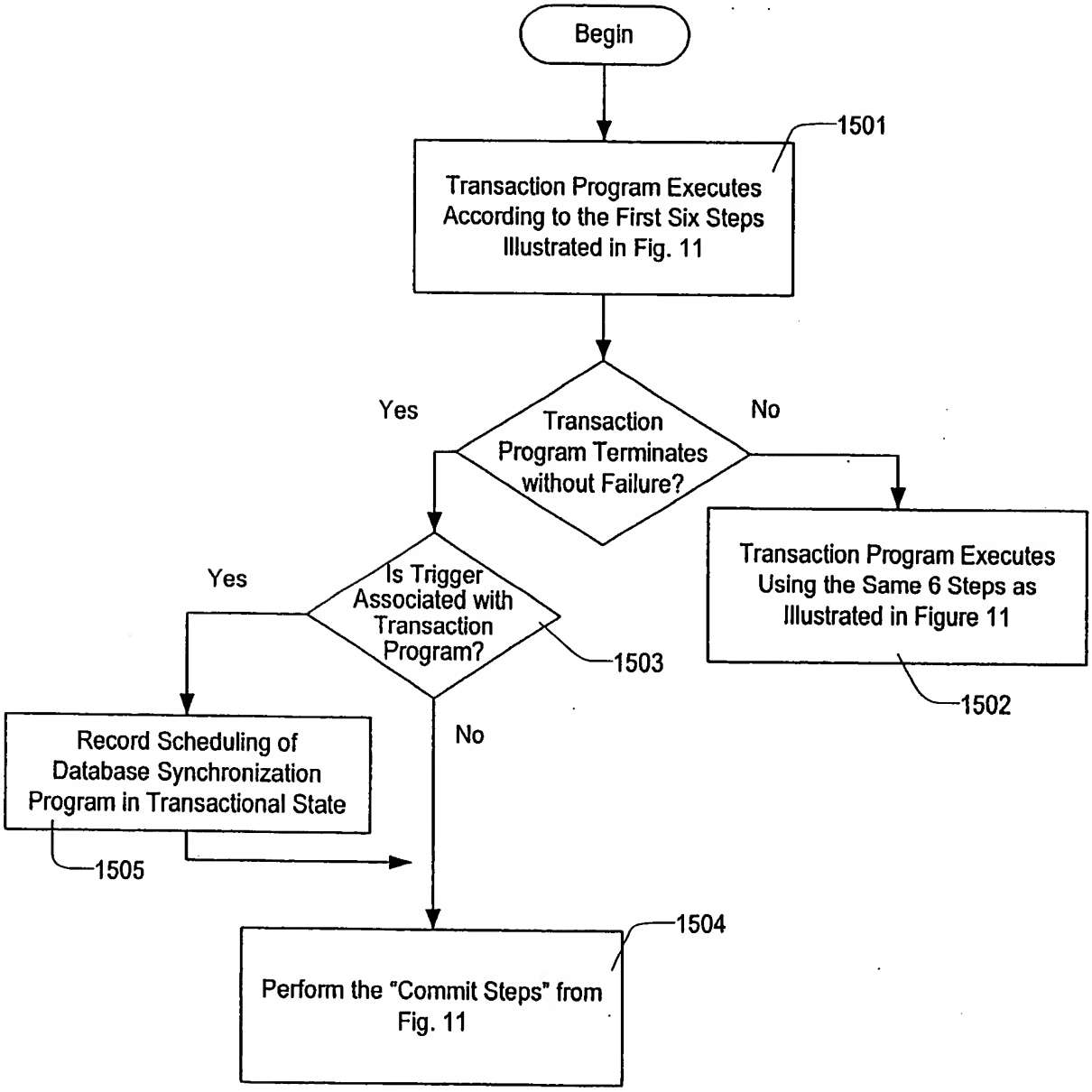

Begin
1501
Transaction Program Executes According to the First Six Steps Illustrated in Fig. 11
Yes
Transaction Program Terminates without Failure?
No
Transaction Program Executes Using the Same 6 Steps as Illustrated in Figure 11
1502
Yes
Is Trigger Associated with Transaction Program?
1503
No
Record Scheduling of Database Synchronization Program in Transactional State
1505
1504
Perform the "Commit Steps" from Fig. 11

FIG. 15

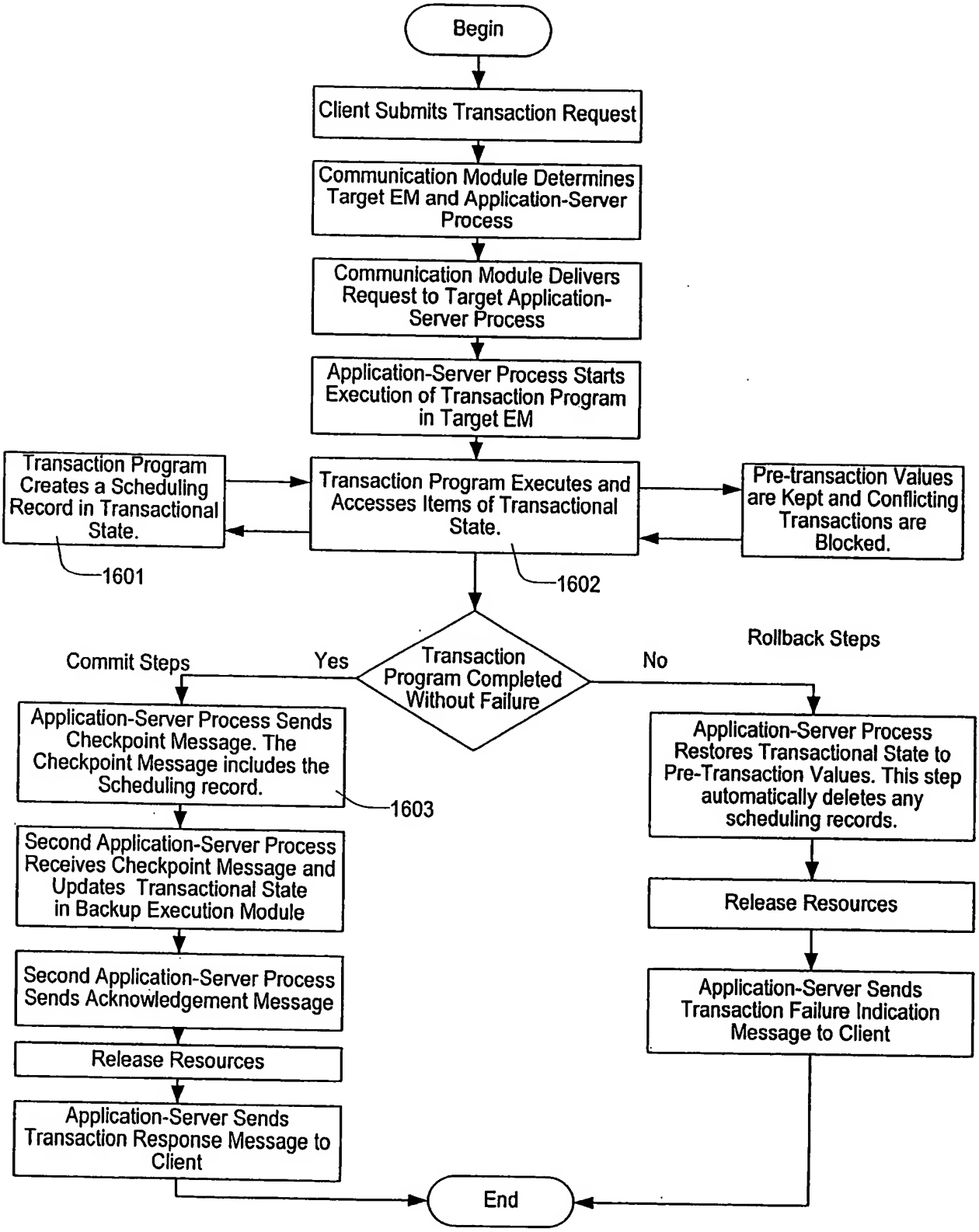

Begin
Client Submits Transaction Request
Communication Module Determines Target EM and Application-Server Process
Communication Module Delivers Request to Target Application-Server Process
Application-Server Process Starts Execution of Transaction Program in Target EM
Transaction Program Creates a Scheduling Record in Transactional State.
1601
Transaction Program Executes and Accesses Items of Transactional State.
1602
Pre-transaction Values are Kept and Conflicting Transactions are Blocked.
Transaction Program Completed Without Failure
Yes
Commit Steps
No
Rollback Steps
Application-Server Process Sends Checkpoint Message. The Checkpoint Message includes the Scheduling record.
1603
Second Application-Server Process Receives Checkpoint Message and Updates Transactional State in Backup Execution Module
Second Application-Server Process Sends Acknowledgement Message
Release Resources
Application-Server Sends Transaction Response Message to Client
Application-Server Process Restores Transactional State to Pre-Transaction Values. This step automatically deletes any scheduling records.
Release Resources
Application-Server Sends Transaction Failure Indication Message to Client
End

FIG. 16

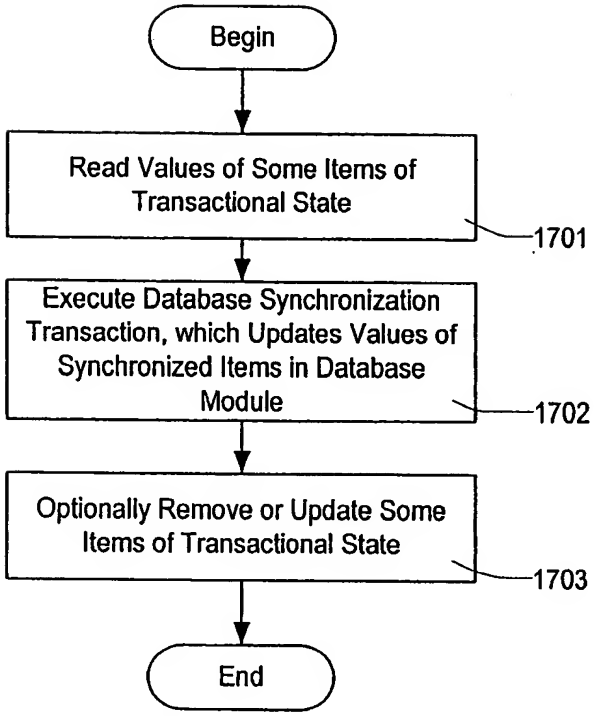

Begin
Read Values of Some Items of Transactional State
1701
Execute Database Synchronization Transaction, which Updates Values of Synchronized Items in Database Module
1702
Optionally Remove or Update Some Items of Transactional State
1703
End

FIG. 17

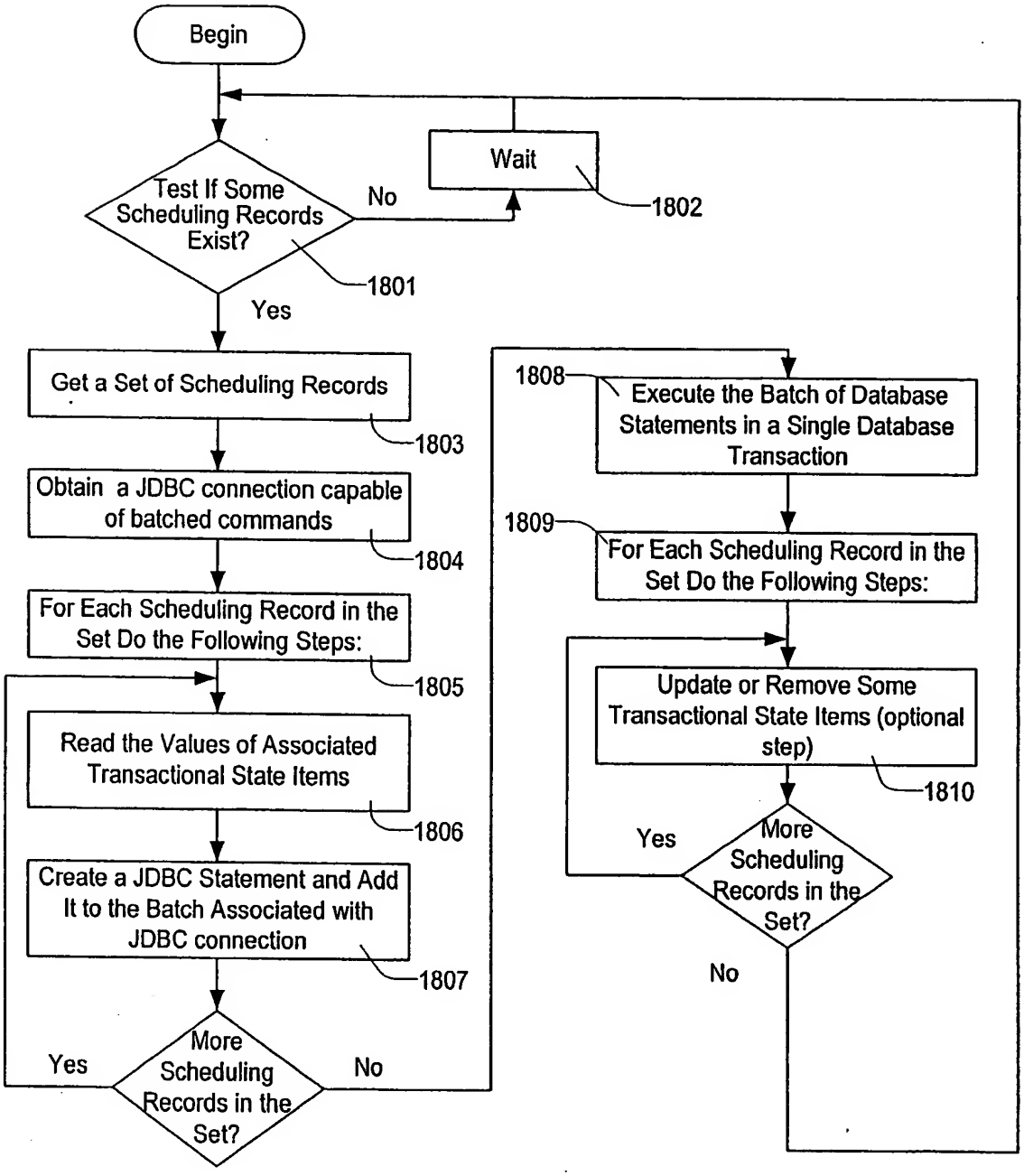
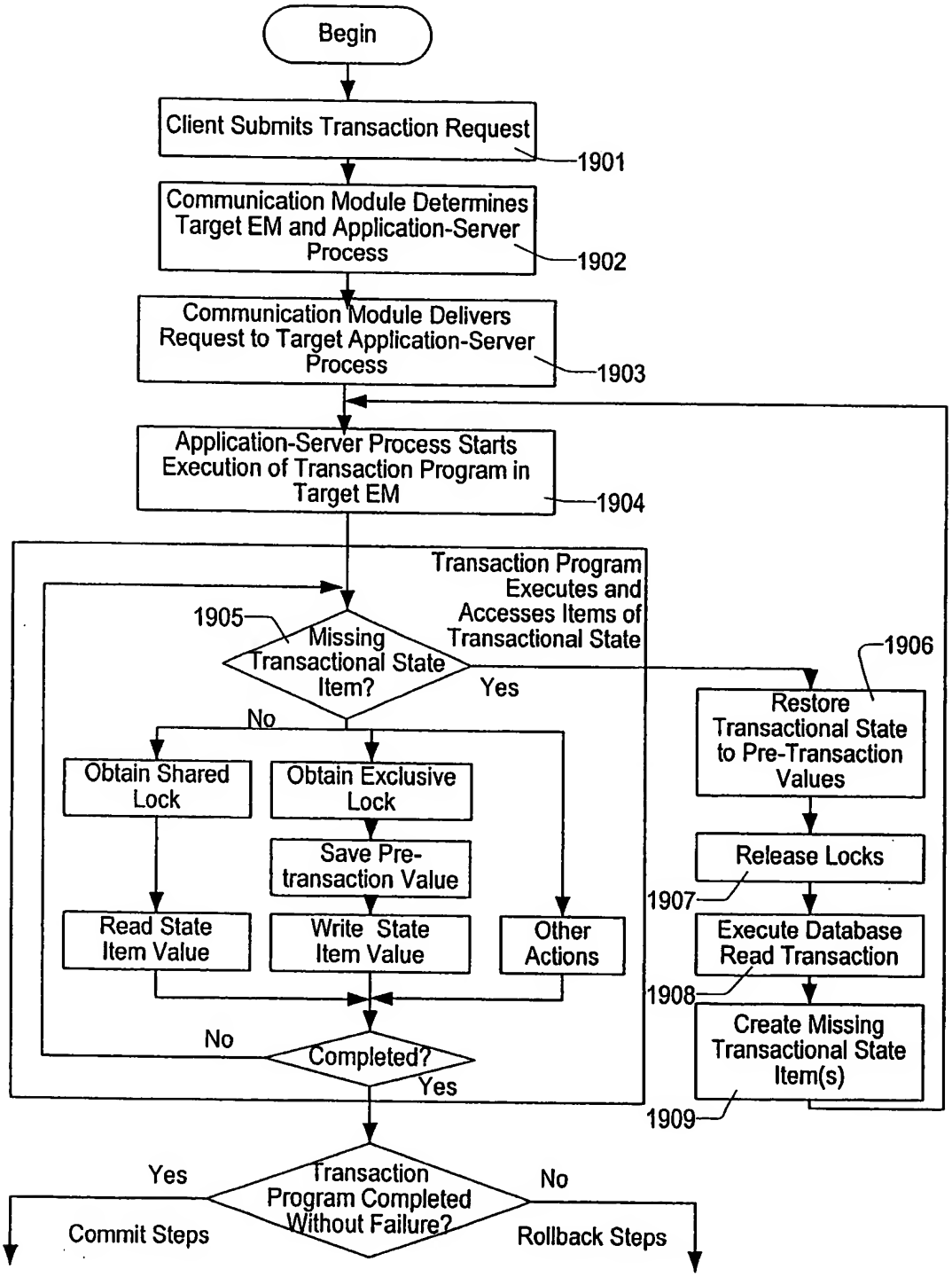

Begin
Test If Some Scheduling Records Exist?
No
Wait
1802
1801
Yes
Get a Set of Scheduling Records
1803
Obtain a JDBC connection capable of batched commands
1804
For Each Scheduling Record in the Set Do the Following Steps:
1805
Read the Values of Associated Transactional State Items
1806
Create a JDBC Statement and Add It to the Batch Associated with JDBC connection
1807
More Scheduling Records in the Set?
Yes
No
1808
Execute the Batch of Database Statements in a Single Database Transaction
1809
For Each Scheduling Record in the Set Do the Following Steps:
Update or Remove Some Transactional State Items (optional step)
1810
More Scheduling Records in the Set?
Yes
No

FIG. 18



Same Steps As in Fig. 12 or Fig. 13

FIG. 19

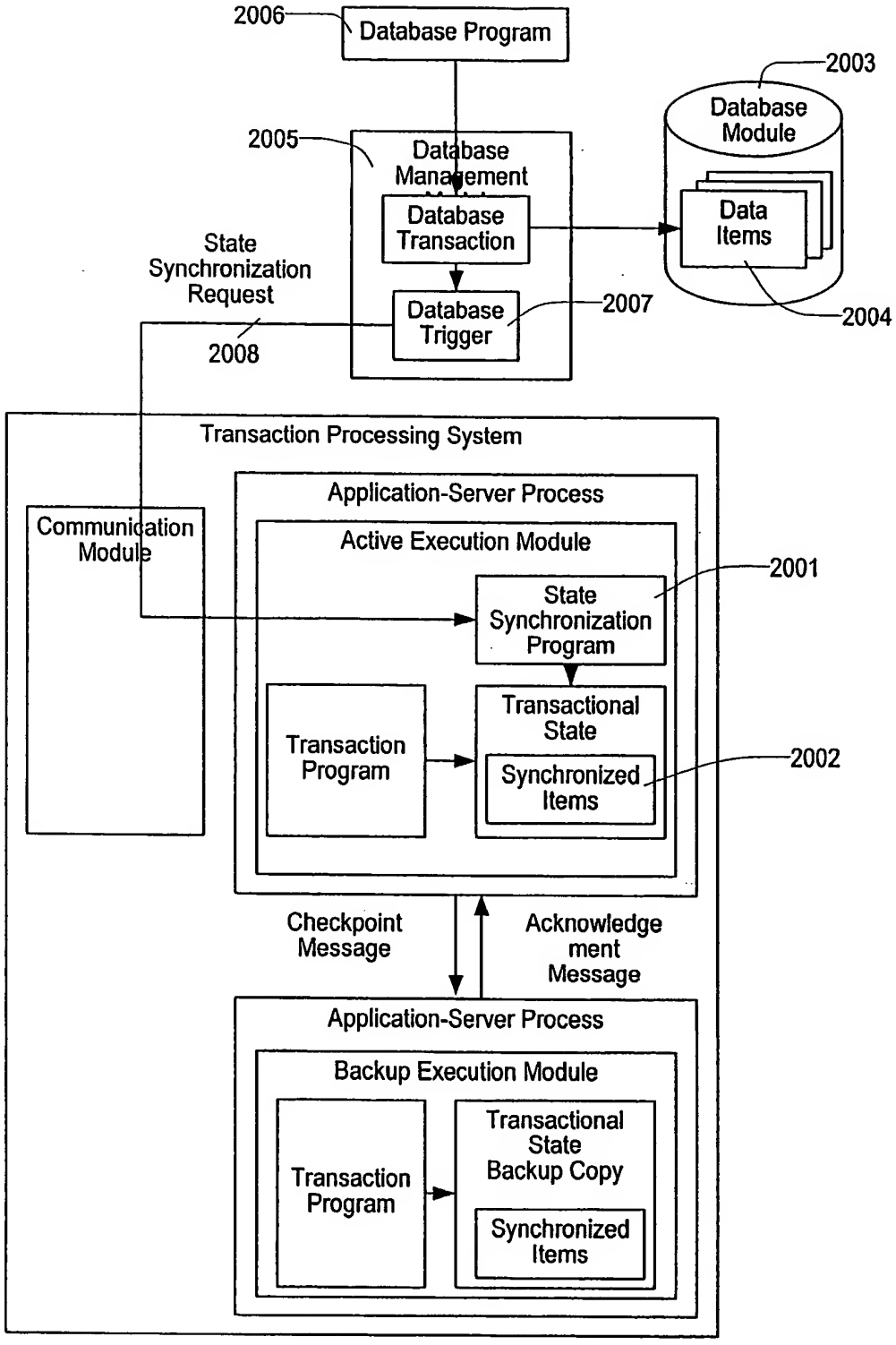

2006
Database Program
2003
Database Module
2005
Database Management
Database Transaction
Data Items
State Synchronization Request
Database Trigger
2007
2004
2008
Transaction Processing System
Application-Server Process
Communication Module
Active Execution Module
2001
State Synchronization Program
Transaction Program
Transactional State
Synchronized Items
2002
Checkpoint Message
Acknowledgement Message
Application-Server Process
Backup Execution Module
Transaction Program
Transactional State Backup Copy
Synchronized Items

FIG. 20

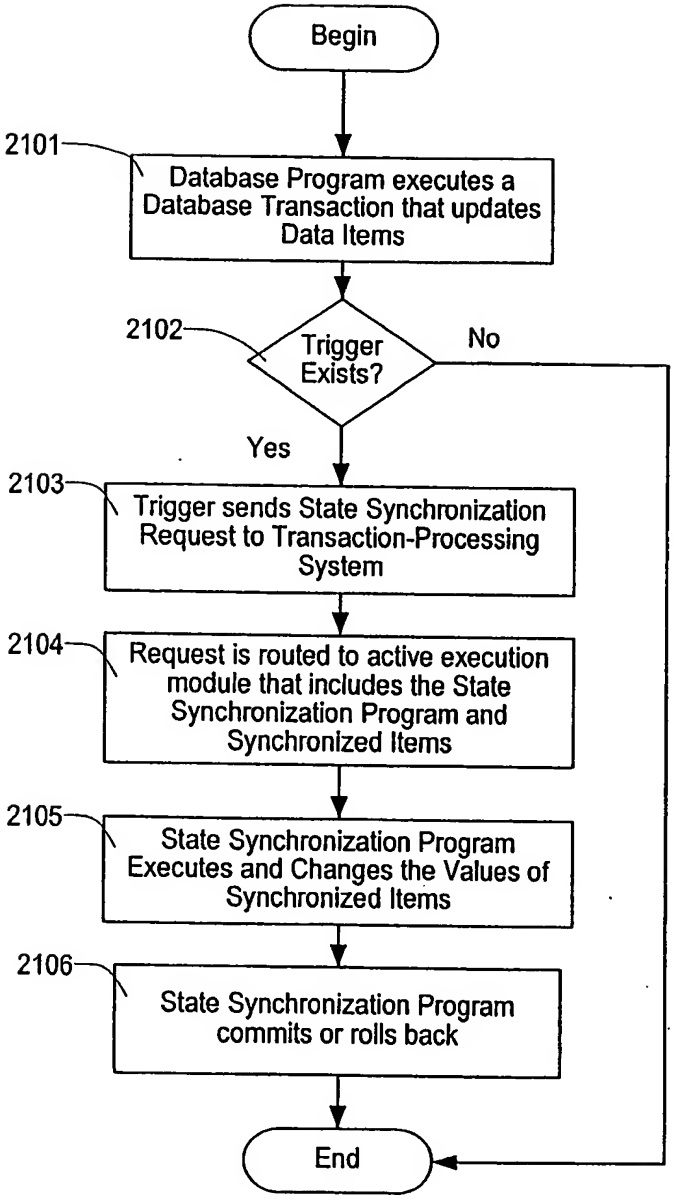

Begin
2101
Database Program executes a Database Transaction that updates Data Items
2102
Trigger Exists?
No
Yes
2103
Trigger sends State Synchronization Request to Transaction-Processing System
2104
Request is routed to active execution module that includes the State Synchronization Program and Synchronized Items
2105
State Synchronization Program Executes and Changes the Values of Synchronized Items
2106
State Synchronization Program commits or rolls back
End

FIG. 21

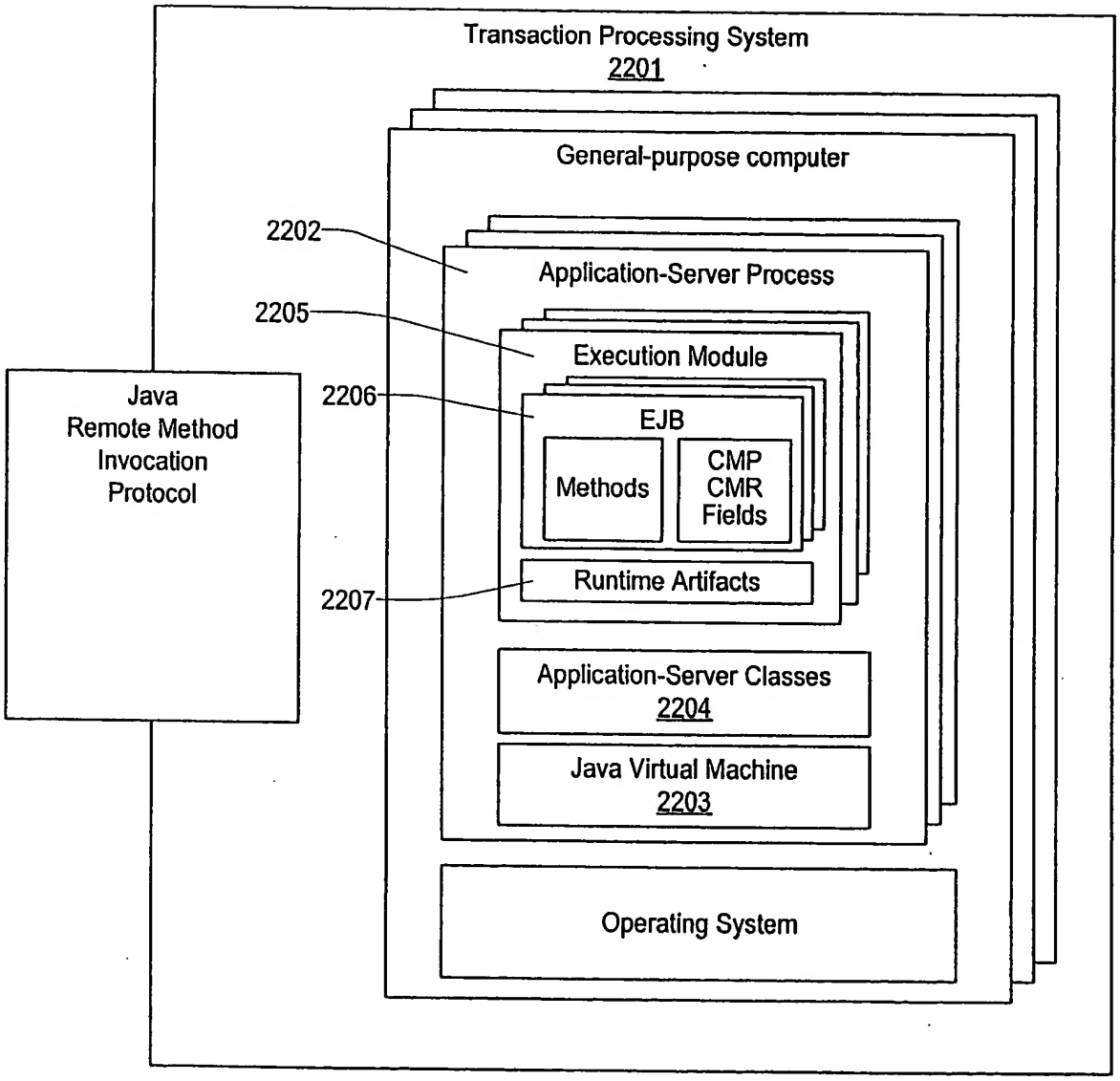

Transaction Processing System
2201
General-purpose computer
2202
Application-Server Process
2205
Execution Module
Java
Remote Method
Invocation
Protocol
2206
EJB
Methods
CMP
CMR
Fields
Runtime Artifacts
2207
Application-Server Classes
2204
Java Virtual Machine
2203
Operating System

FIG. 22

```
public abstract class AccountBean implements EntityBean {
   public abstract String getAccountNumber();
   public abstract void setAccountNumber();

   public abstract double getBalance();
   public abstract void setBalance(double newBalance);

   public void debit(double amount) {
      setBalance(getBalance() - amount);
   }

   public void credit(double amount) {
      setBalance(getBalance() + amount);
   }

   // ...
}
```

FIG. 23

```
class AccountBeanImpl extends AccountBean {

   private double _balance;
   private double _balance_pretx;
   private boolean _balance_modified;

   public double getBalance() {
      return _balance;
   }

   public void setBalance(double newBalance) {
      if (!_balance_modified) {
         _balance_modified = true;
         _balance_pretx = _balance;
      }
      _balance = newBalance;
   }

   // ... similar code for accountNumber
}
```

FIG. 24

```
public class Account {
   private String accountNumber;
   private double balance;

   public void debit(double amount) {
      balance -= amount;
   }

   public void credit(double amount) {
      balance += amount;
   }
}
```

FIG. 25

```
public interface Account extends EJBLocalObject {

   // CMP field accountNumber
   public String getAccountNumber();

   // CMP field balance
   public double getBalance();
   public void setBalance(double newBalance);

   // EJB business methods.
   public void debit(double amount);
   public void credit(double amount);
}
```

FIG. 26

```
public abstract class AccountBeanTrigger extends AccountBean {
   AccountSynchronizationHome accountSynchronizationHome;

   public void debit(double amount) throws EJBException {
      super.debit(amount);
      try {
         accountSynchronizationHome.create(getAccountNumber());
      } catch (DuplicateKeyException e) {
         // This exception indicates that a database
          // synchronization program for this
         // account number is already scheduled.
      } catch (Exception e) {
         // Unexpected error.
         throw new EJBException(e);
      }
   }
}
```

FIG. 27

```
public class AccountBeanImpl implements AccountBeanTrigger {
  // same code as in Fig. 24
}
```

FIG. 28

```
public interface DatabaseSynchronization extends EJBLocalObject {
   void readValues() throws Exception;
   void updateDatabase() throws Exception;
   void removeItems() throws Exception;
}
```

FIG. 29

```
public interface AccountSynchronization extends DatabaseSynchronization {
}
```

FIG. 30

```
public interface AccountSynchronizationHome extends EJBLocalHome {
   AccountSynchronization create(String accountNumber)
         throws CreateException;
   AccountSynchronization findByPrimaryKey(String accountNumber)
         throws FinderException;
}
```

FIG. 31

```
public abstract class AccountSynchronizationBean implements EntityBean {

   EntityContext entityContext;
   AccountHome accountHome;

   public abstract String getAccountNumber();
   public abstract void setAccountNumber(String accountNumber);

   private Account account;

   // Values of read fields.
   private String accountNumber;
   private double latestBalance;

   public String ejbCreate(String accountNumber) {
      setAccountNumber(accountNumber);
      account = accountHome.findByPrimaryKey(accountNumber);
      return null;
   }

   public void readValues() throws Exception {
      accountNumber = getAccountNumber();
      latestBalance = account.getBalance();
   }

   public void updateDatabase() throws Exception {
      Connection conn = ...; // obtain a JDBC connection;
      PreparedStatement st = conn.prepareStatement(
            "UPDATE Account SET balance = ? WHERE accountNumber = ?");

      st.setDouble(1, latestBalance);
      st.setString(2, accountNumber);

      if (st.executeUpdate() < 1) {
         throw new Exception("Account " + accountNumber + " does not exist");
      }
   }

   public void removeItems() throws Exception {
      if (account.getBalance() == latestBalance) {
         entityContext.getEJBLocalObject().remove();
      } else {
         // Retain this EJB because another synchronization transaction
         // is necessary.
      }
   }

   // ...
}
```

FIG. 32

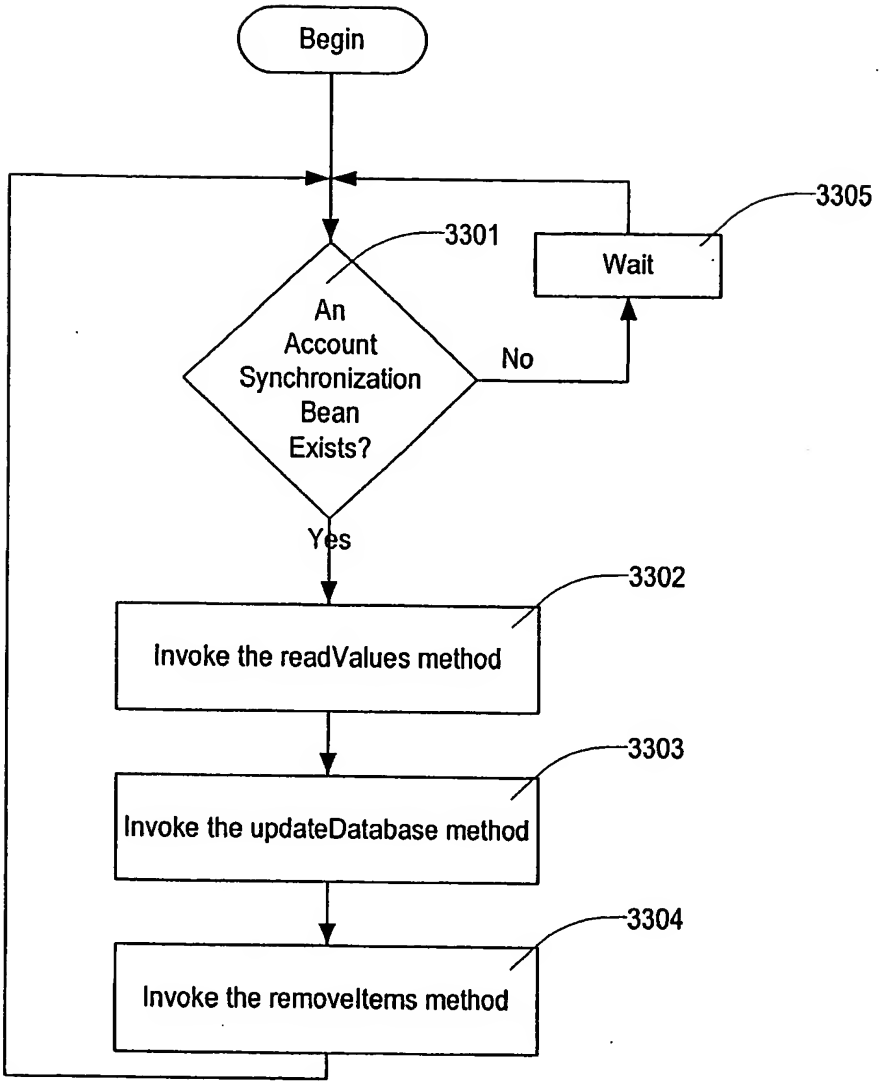

Begin
3301
An Account Synchronization Bean Exists?
No
Wait
3305
Yes
3302
Invoke the readValues method
3303
Invoke the updateDatabase method
3304
Invoke the removeItems method

FIG. 33

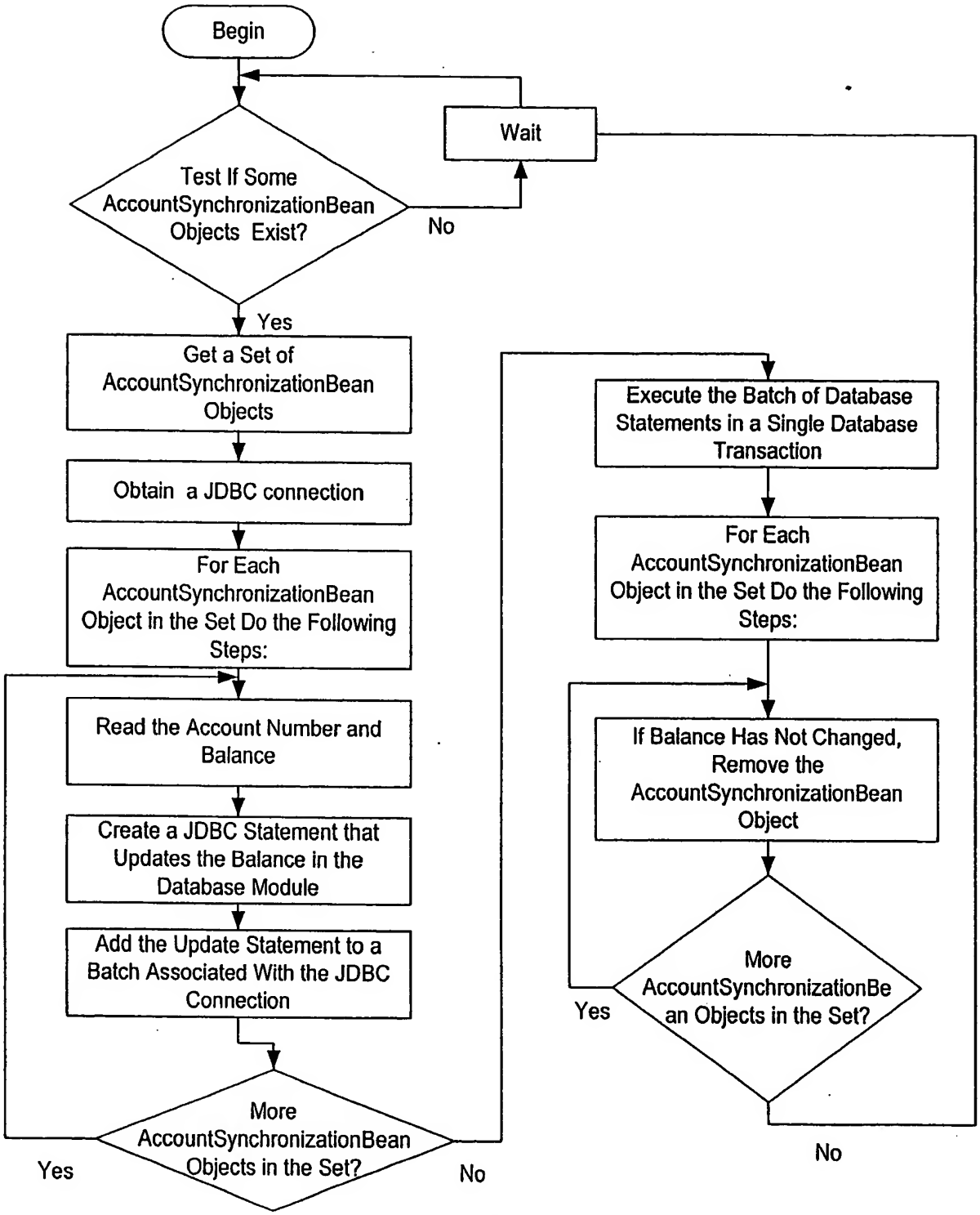

Begin
Wait
Test If Some AccountSynchronizationBean Objects Exist?
No
Yes
Get a Set of AccountSynchronizationBean Objects
Obtain a JDBC connection
For Each AccountSynchronizationBean Object in the Set Do the Following Steps:
Read the Account Number and Balance
Create a JDBC Statement that Updates the Balance in the Database Module
Add the Update Statement to a Batch Associated With the JDBC Connection
More AccountSynchronizationBean Objects in the Set?
Yes
No
Execute the Batch of Database Statements in a Single Database Transaction
For Each AccountSynchronizationBean Object in the Set Do the Following Steps:
If Balance Has Not Changed, Remove the AccountSynchronizationBean Object
More AccountSynchronizationBean Objects in the Set?
Yes
No

FIG. 34

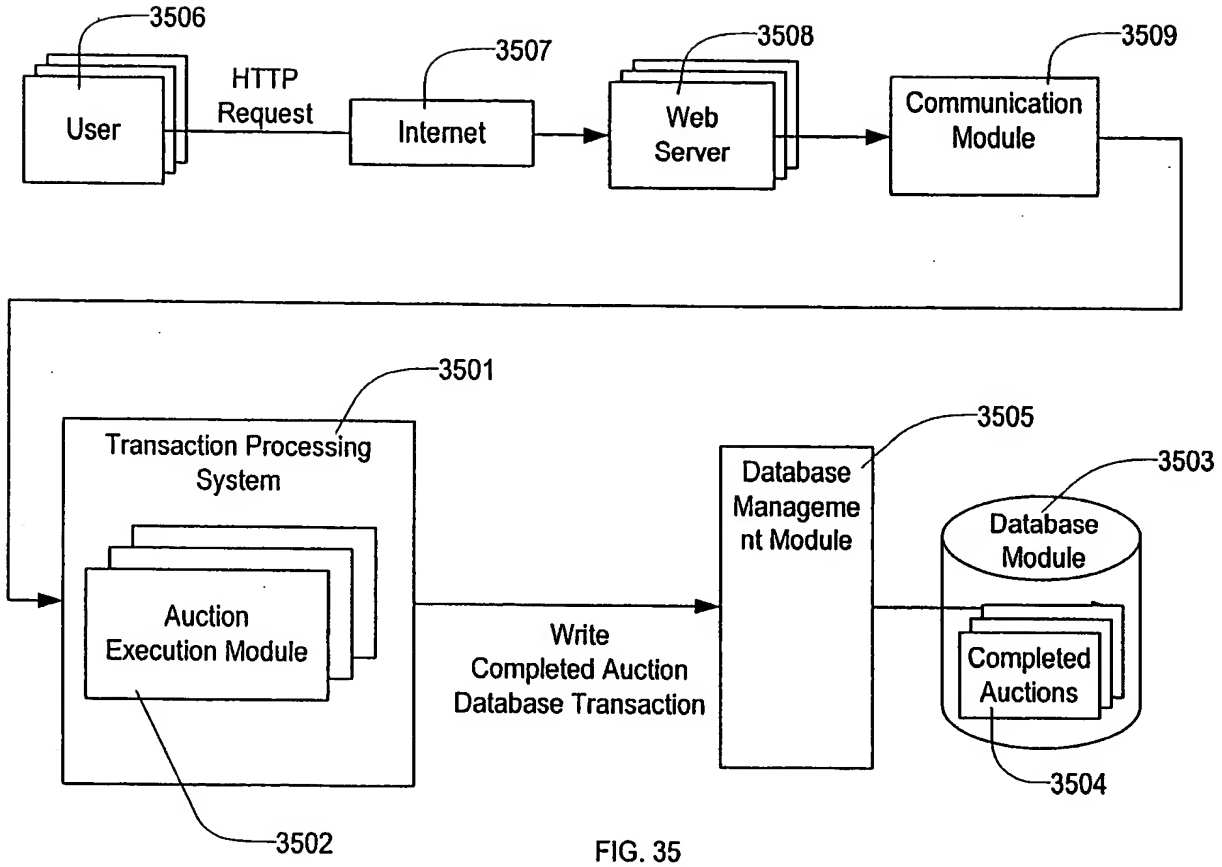

3506
User
HTTP
Request
3507
Internet
3508
Web
Server
3509
Communication
Module
3501
Transaction Processing
System
Auction
Execution Module
3502
Write
Completed Auction
Database Transaction
3505
Database
Manageme
nt Module
3503
Database
Module
Completed
Auctions
3504

FIG. 35

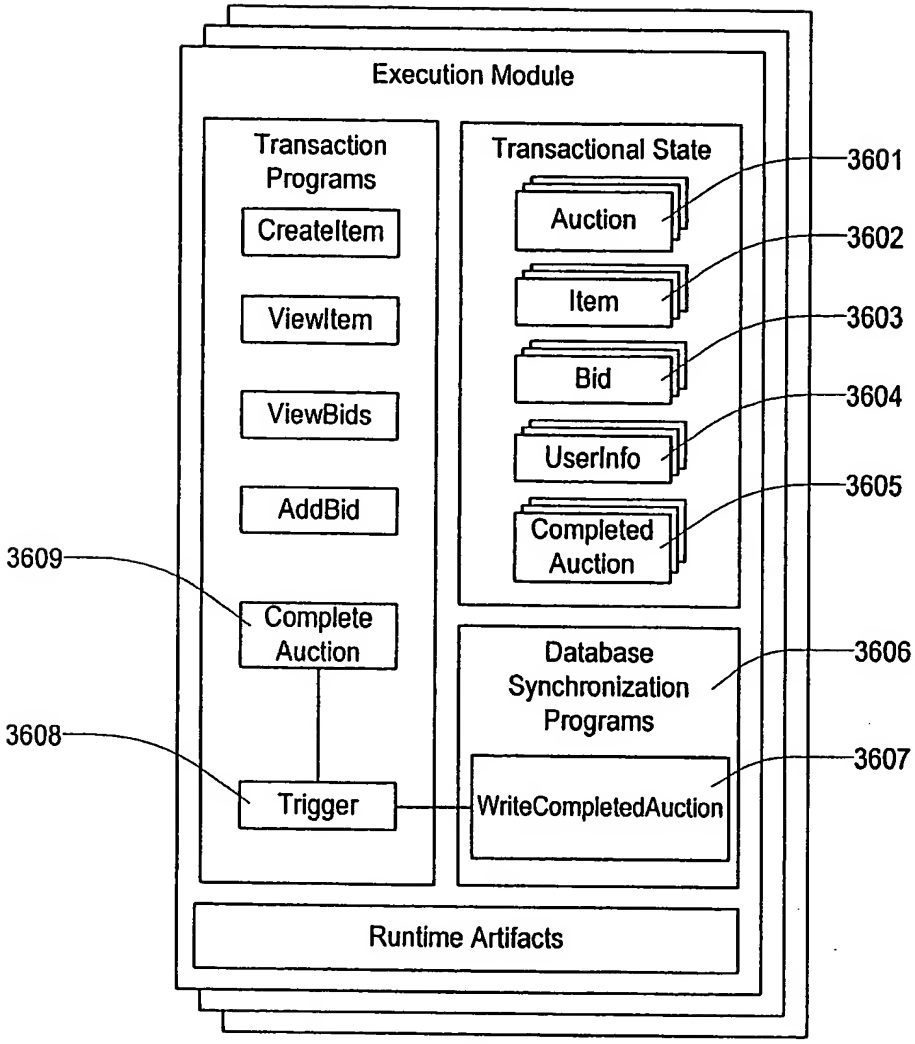

Execution Module
Transaction Programs
CreateItem
ViewItem
ViewBids
AddBid
Complete Auction
Trigger
Transactional State
Auction
Item
Bid
UserInfo
Completed Auction
Database Synchronization Programs
WriteCompletedAuction
Runtime Artifacts
3601
3602
3603
3604
3605
3606
3607
3608
3609

FIG. 36

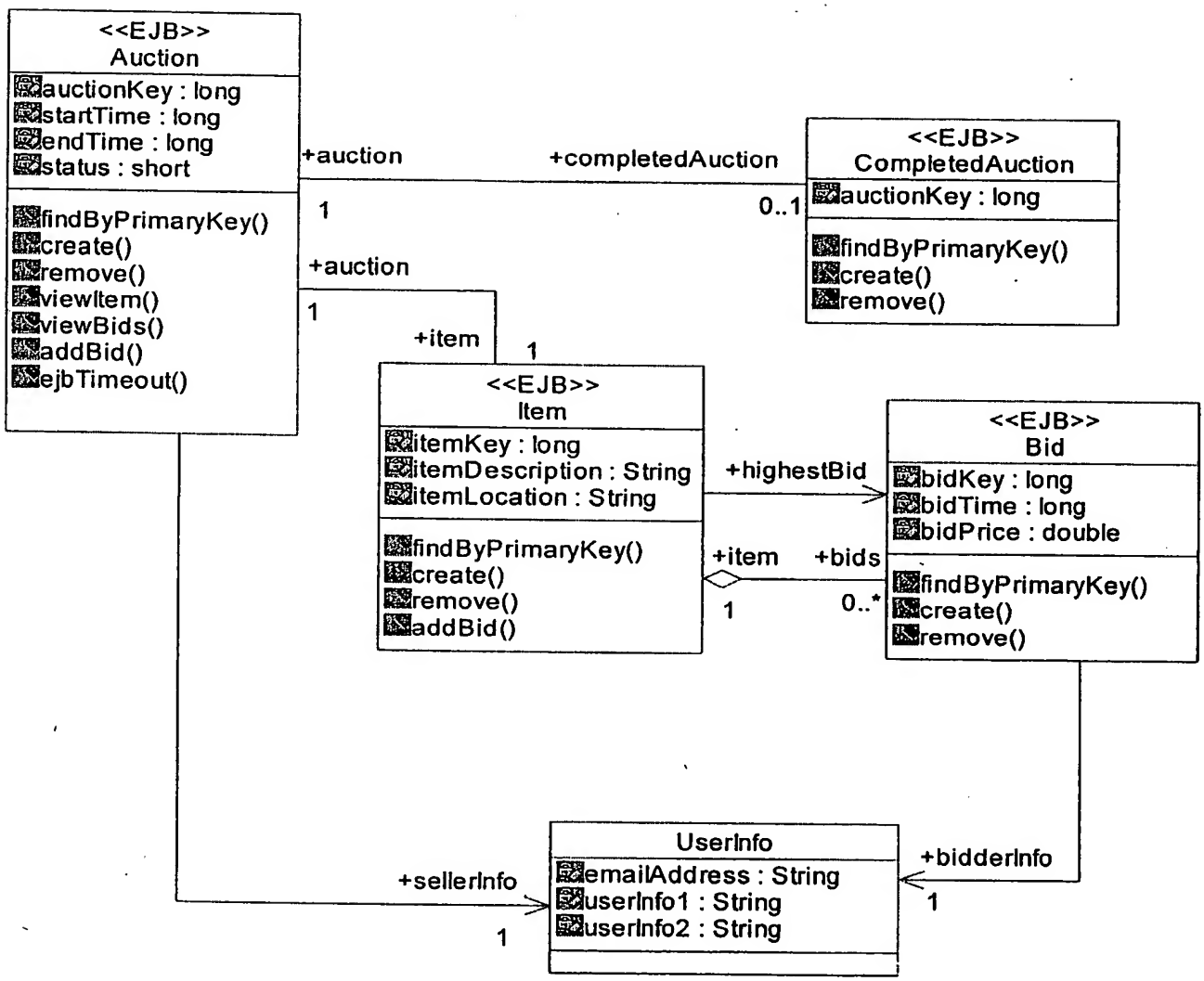

<<EJB>>
Auction
auctionKey : long
startTime : long
endTime : long
status : short
findByPrimaryKey()
create()
remove()
viewItem()
viewBids()
addBid()
ejbTimeout()
+auction
1
+completedAuction
0..1
<<EJB>>
CompletedAuction
auctionKey : long
findByPrimaryKey()
create()
remove()
+auction
1
+item
1
<<EJB>>
Item
itemKey : long
itemDescription : String
itemLocation : String
findByPrimaryKey()
create()
remove()
addBid()
+highestBid
<<EJB>>
Bid
bidKey : long
bidTime : long
bidPrice : double
findByPrimaryKey()
create()
remove()
+item
1
+bids
0..*
UserInfo
emailAddress : String
userInfo1 : String
userInfo2 : String
+sellerInfo
1
+bidderInfo
1

FIG. 37

```
/**
 * Enterprise JavaBean implementing an auction.
 */
abstract public class AuctionBean implements EntityBean, TimedObject {
   EntityContext entityContext;
   UserHome userHome;
   ItemHome itemHome;
   // EJB CMP and CMR fields accessor methods
   public abstract void setAuctionKey(Long auctionKey);
   public abstract Long getAuctionKey();
   public abstract void setItem(Item item);
   public abstract void setStartTime(long startTime);
   public abstract void setEndTime(long endTime);
   public abstract void setStatus(short status);
   public abstract void setCompletedAuction(CompletedAuction completedAuction);
   public abstract void setSellerInfo(UserInfo sellerInfo);
   public abstract short getStatus();
   public abstract CompletedAuction getCompletedAuction();
   public abstract UserInfo getSellerInfo();
   public abstract long getEndTime();
   public abstract long getStartTime();
   public abstract Item getItem();
   public void ejbLoad() {}
   public void ejbStore() {}
   public void ejbActivate() {}
   public void ejbPassivate() {}
   public void unsetEntityContext() { entityContext = null;}
   public void setEntityContext(EntityContext entityContext) {
      this.entityContext = entityContext;
      userHome = ...;
      itemHome = ...;
   }

   /** Creates an auction for an item. */
   public Long ejbCreate(UserInfo sellerInfo, CreateItemInfo itemCopy, CategoryRemote category)
                  throws CreateException, AuctionException {
      setAuctionKey(new Long(KeyGen.getNextKey()));
      setStartTime(itemCopy.getStartTime());
      setEndTime(itemCopy.getEndTime());
      setSellerInfo(sellerInfo);
      itemCopy.setCategory(category);
      return null;
   }

public void ejbPostCreate(UserInfo sellerInfo, CreateItemInfo itemCopy, CategoryRemote category)
                  throws CreateException, AuctionException {
      try {
         itemHome.create(new Long(KeyGen.getNextKey()),
                  (Auction) entityContext.getEJBLocalObject(), itemCopy);
      } catch (CreateException ex) {
         entityContext.setRollbackOnly();
         throw ex;
      }
```

FIG. 38 Code continues in Fig. 39

Continuation of code in Fig. 38

```
        TimerService timerService = entityContext.getTimerService();
        long endTime = itemCopy.getEndTime();
        long currentTime = System.currentTimeMillis();
        long duration =  endTime - currentTime;
        timerService.createTimer(duration, null);
    }

    /** Removes this auctioned item. */
    public void ejbRemove() throws RemoveException {
        getItem().remove();
    }

    public double getHighestBid() throws AuctionException {
            return getItem().getHighestBid();
    }

    /** Obtains the current information about this auctioned item. */
    public ViewItemResult viewItem() throws AuctionException {
        Item item = getItem();
        ViewItemResult copy = new ViewItemResult();
        copy.setDescription(item.getItemDescription());
        copy.setStartTime(getStartTime());
        copy.setEndTime(getEndTime());
        copy.setLocation(item.getItemLocation());
        copy.setItemKey(getAuctionKey().longValue());
        copy.setHighestBid(item.getHighestBid());
        copy.setNumberOfBids(item.getBids().size());
        return copy;
    }

    /** Adds a bid on this auctioned item. */
    public void addBid(UserInfo bidderInfo, double bid) throws AuctionException {
        getItem().addBid(bidderInfo, bid);
    }

    /** Obtains all bids on this auctioned item. */
    public ViewBidsResult viewBids() throws AuctionException {
        return new ViewBidsResult(getItem().getBids());
    }

    /** Handles the end of auction timer invocation. */
    public void ejbTimeout(javax.ejb.Timer timer) {
        try {
            LocalHomes.getCompletedAuctionHome().create((Auction) entityContext.getEJBLocalObject());
        } catch (CreateException e) {
            logger.log(Level.WARNING, "unexpected CreateException", e);
        }
    }
}
```

FIG. 39

```
/**
 * Background thread that periodically writes completed auctions to a database.
 */
public class DatabaseUpdateThread extends Thread {
   static final int maxRowsPerInsert = 64; // Maximum batch size
   UserTransaction ut = ...;                    // Obtain the UserTransaction interface
   CompletedAuctionHome h = ...;          // Obtain EJB local home interface

   public void run() {
      while (true) {
            try {
               Thread.sleep(1000);// Sleep 1 second
               writeCompletedAuctions();
            } catch (InterruptedException e) {}
         }
      }
   }

   private void writeCompletedAuctions() {
      CompletedAuctionHome h = LocalHomes.getCompletedAuctionHome();
      ArrayList keyArrayList;
      // Obtain a ArrayList with a copy of all completed auction keys.
      try {
         ut.begin();                    // This an EJB, not JDBC, transaction
         keyArrayList = new ArrayList(h.obtainAllKeys());
         ut.commit();
         txInProgress = false;
      } catch (Exception e) {
          // Handle exceptions
      }

      // If keyArrayList is too big, break the database updates into multiple smaller batches.
      int totalCount = keyArrayList.size();
      int processedCount = 0;
      while (processedCount < totalCount) {
         int thisBatchCount = Math.min(maxRowsPerInsert, totalCount - processedCount);
         processBatch(keyArrayList.subList(processedCount, processedCount + thisBatchCount));
         processedCount += thisBatchCount;
      }
   }

  /** Executes a batch using a single database transaction. */
   private void processBatch(List keyArrayList) {
      Connection con = null;    Code continues in Fig. 41
      PreparedStatement stmt = null;
      SQLException sqlException = null;
      try {
         con = ...; // Obtain a JDBC connection
```

FIG. 40 Code continues in Fig. 41

Continuation of code in Fig. 40

```
        // Read all completed transactions in this Auction EM in one read-only EJB transaction.
        stmt = con.prepareStatement("INSERT INTO COMPLETED_AUCTIONS VALUES(?,?,?,?,?,?,?,?,?)");
        try {
          ut.begin();                          // This is an EJB, not JDBC, transaction
          for (Iterator it = keyArrayList.iterator(); it.hasNext(); ) {
            Long key = (Long) it.next();
            CompletedAuction completedAuction = h.findByPrimaryKey(key);
            Auction auction = completedAuction.getAuction();
            Item item = auction.getItem();
            UserInfo sellerInfo = auction.getSellerInfo();
           UserInfo buyerInfo = item.getHighestBid().getBidderInfo();

           stmt.setLong(1, item.getItemKey());              // Item key
            stmt.setString(2, item.getItemDescription());          // Item description
            stmt.setString(3, item.getItemLocation());             // Item location
            stmt.setTimestamp(4, new Timestamp(auction.getStartTime())); // Auction start time
            stmt.setTimestamp(5, new Timestamp(auction.getEndTime()));  // Auction end time
            stmt.setString(6, sellerInfo.getEmailAddress());       // Seller's key
            stmt.setString(7, buyerInfo.getEmailAddress();         // Buyer's key
            stmt.setDouble(8, item.getHighestBid());               // Final price

            stmt.addBatch();
          }
          ut.commit();
        } catch (Exception e) {
          // Handle exception
        }

        // Execute the database insert batch outside of any EJB transaction to
        // to prevent holding EJB locks across database access.
        stmt.executeBatch();
        con.commit();

        // Release database connection here
      } catch (SQLException e) {
        // Handle exception
      }

      /* Remove all CompletedAuctionBean objects that have been written to the database. */
      try {
        ut.begin();
        for (Iterator it = keyArrayList.iterator(); it.hasNext(); ) {
            Long key = (Long) it.next();
            CompletedAuction completedAuction = h.findByPrimaryKey(key);
            completedAuction.remove();
        }
        ut.commit();
      } catch (Exception e) {
        // Handle exception
      }
    }
}
```

FIG. 41

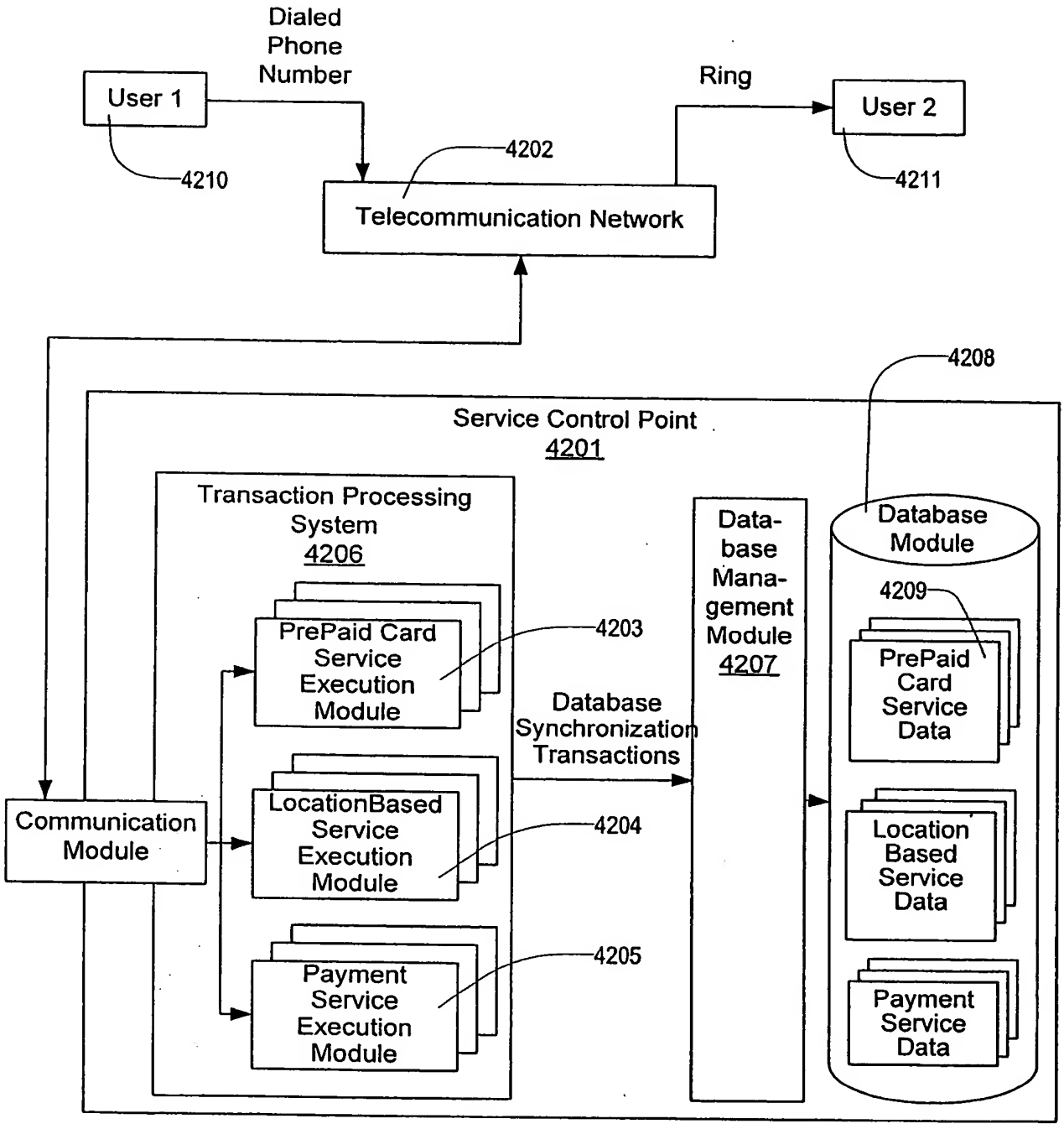

Dialed
Phone
Number
User 1
4210
4202
Telecommunication Network
Ring
User 2
4211
4208
Service Control Point
4201
Transaction Processing
System
4206
PrePaid Card
Service
Execution
Module
4203
LocationBased
Service
Execution
Module
4204
Payment
Service
Execution
Module
4205
Communication
Module
Database
Synchronization
Transactions
Data-
base
Mana-
gement
Module
4207
Database
Module
4209
PrePaid
Card
Service
Data
Location
Based
Service
Data
Payment
Service
Data

FIG. 42

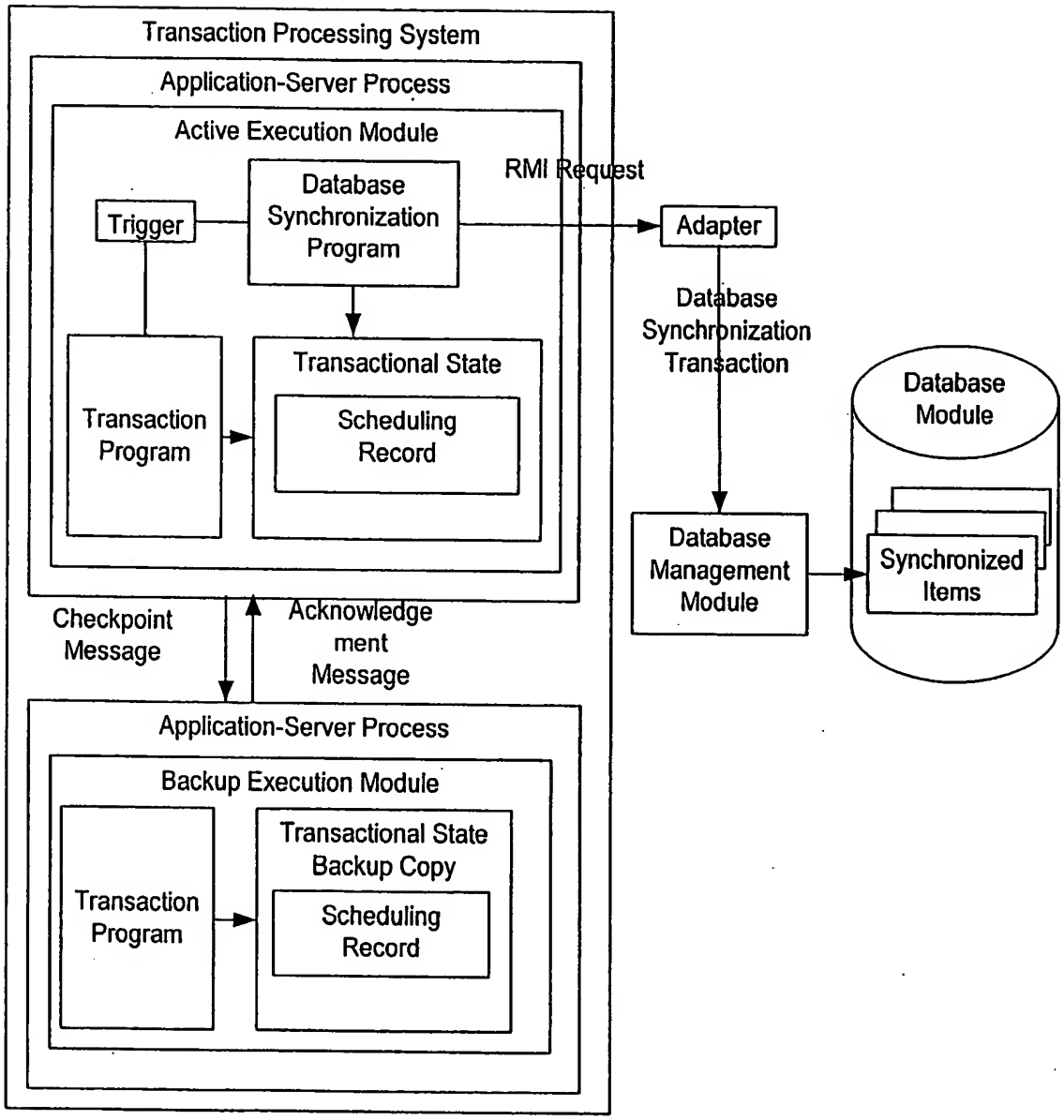

Transaction Processing System
Application-Server Process
Active Execution Module
Trigger
Database Synchronization Program
RMI Request
Adapter
Database Synchronization Transaction
Transaction Program
Transactional State
Scheduling Record
Database Module
Database Management Module
Synchronized Items
Checkpoint Message
Acknowledge ment Message
Application-Server Process
Backup Execution Module
Transaction Program
Transactional State Backup Copy
Scheduling Record

FIG. 43

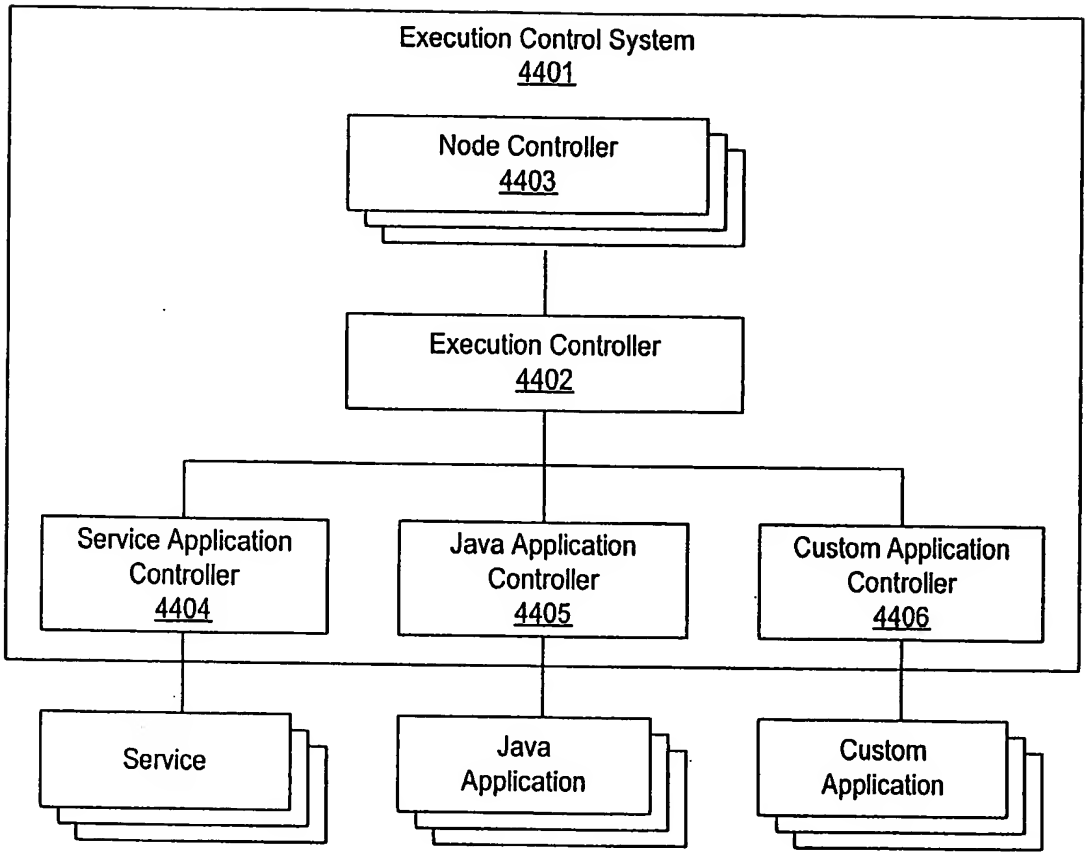

Execution Control System
4401
Node Controller
4403
Execution Controller
4402
Service Application Controller
4404
Java Application Controller
4405
Custom Application Controller
4406
Service
Java Application
Custom Application

Fig 44

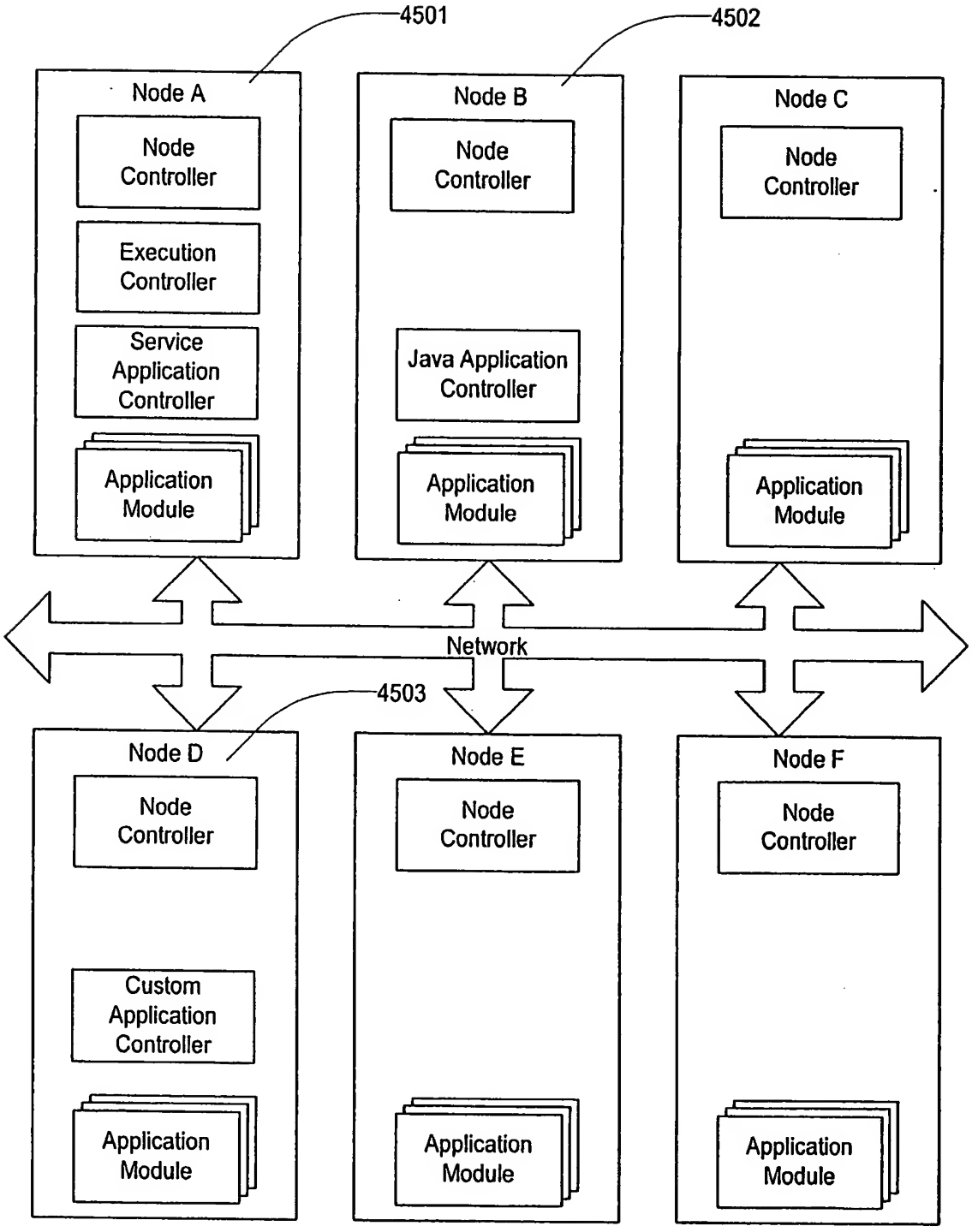

4501
4502
Node A
Node Controller
Execution Controller
Service Application Controller
Application Module
Node B
Node Controller
Java Application Controller
Application Module
Node C
Node Controller
Application Module
Network
4503
Node D
Node Controller
Custom Application Controller
Application Module
Node E
Node Controller
Application Module
Node F
Node Controller
Application Module

Fig 45

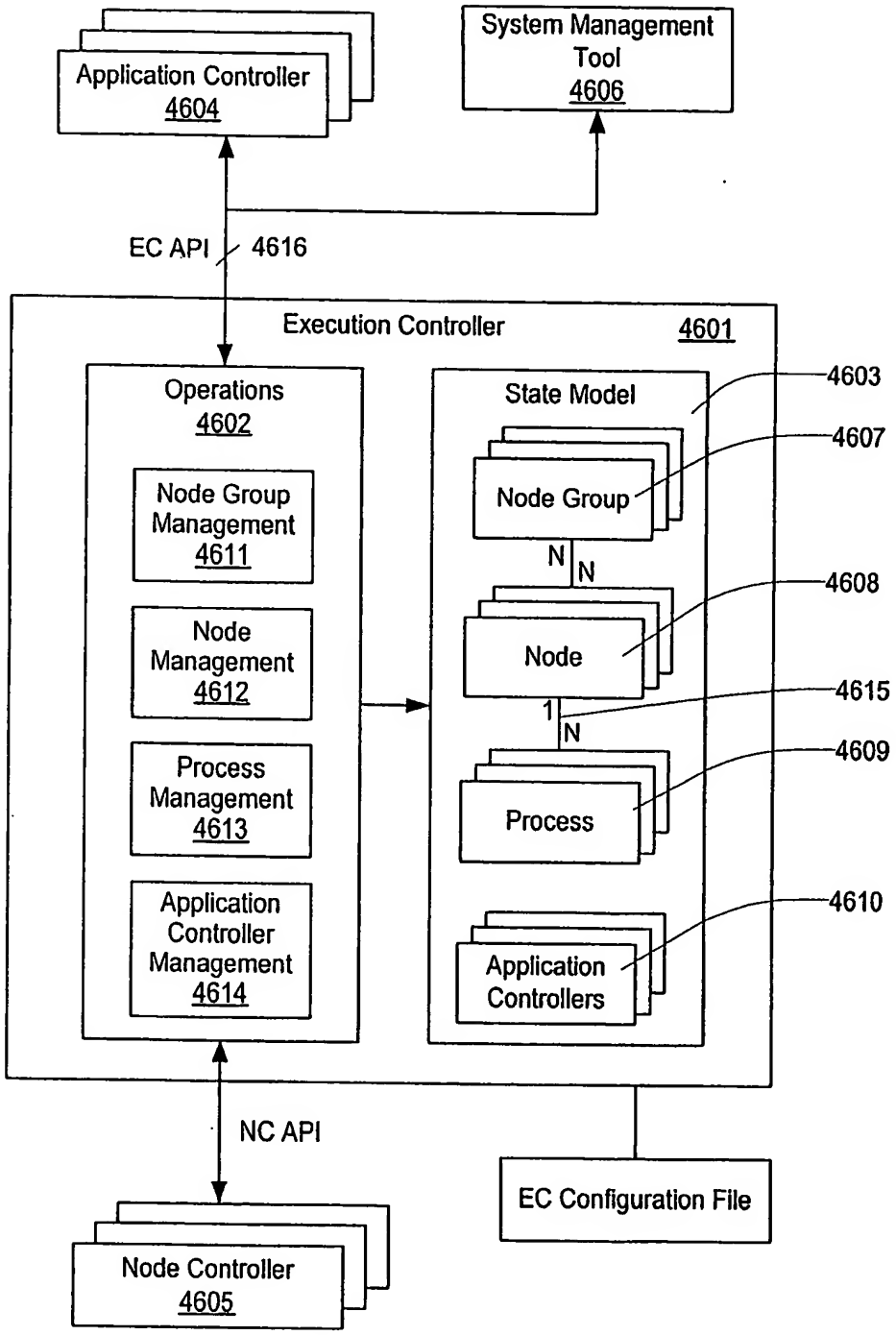

Application Controller
4604
System Management
Tool
4606
EC API
4616
Execution Controller
4601
Operations
4602
Node Group
Management
4611
Node
Management
4612
Process
Management
4613
Application
Controller
Management
4614
State Model
4603
Node Group
4607
N
N
Node
4608
1
N
Process
4609
Application
Controllers
4610
4615
NC API
Node Controller
4605
EC Configuration File

Fig 46

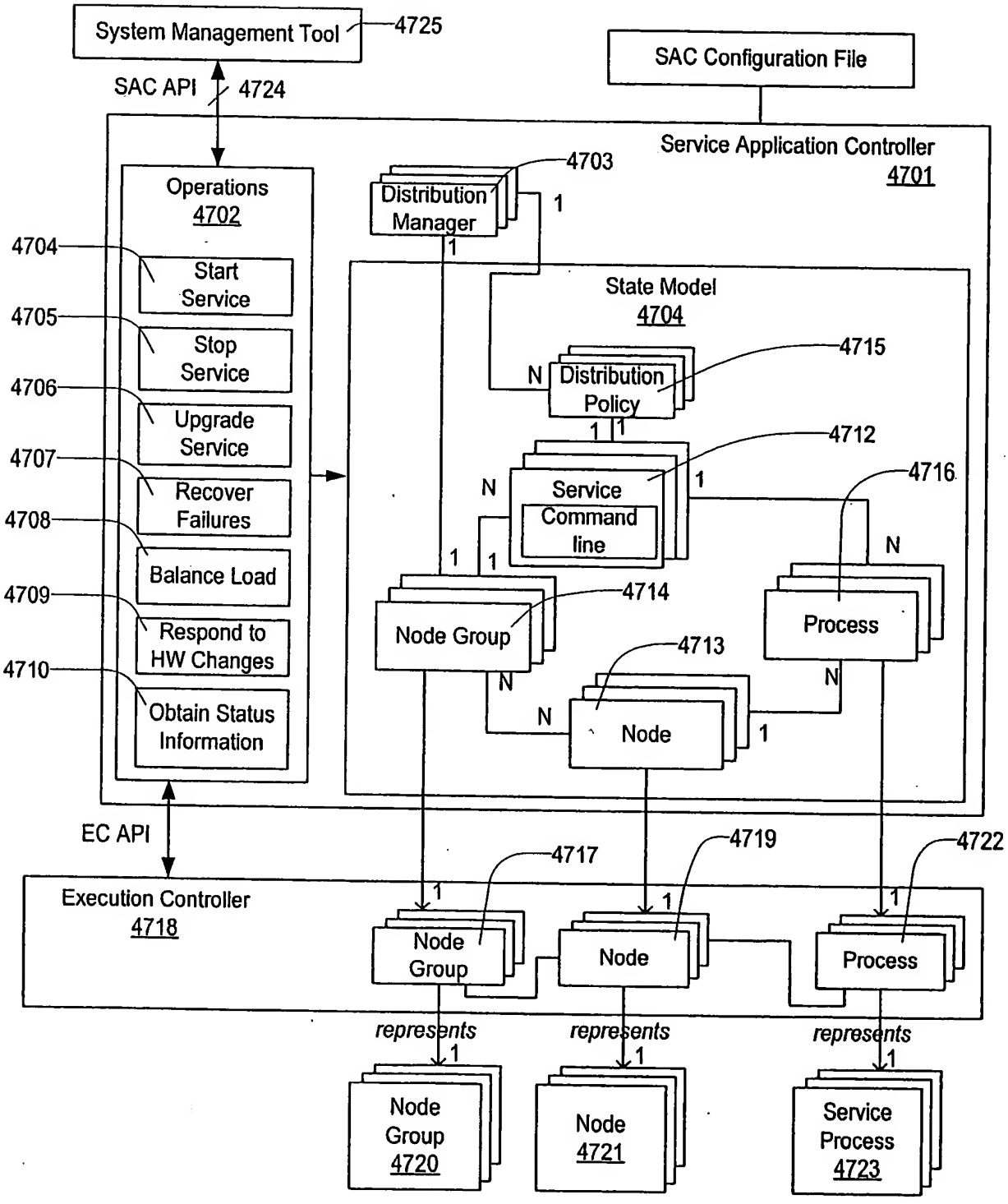

System Management Tool
4725
SAC API
4724
SAC Configuration File
Service Application Controller
4701
Operations
4702
4703
Distribution Manager
1
1
4704
Start Service
4705
Stop Service
4706
Upgrade Service
4707
Recover Failures
4708
Balance Load
4709
Respond to HW Changes
4710
Obtain Status Information
State Model
4704
4715
N
Distribution Policy
1
1
4712
N
Service
Command line
1
4716
1
1
N
Node Group
4714
Process
4713
N
N
N
Node
1
EC API
4717
4719
4722
Execution Controller
4718
1
1
1
Node Group
Node
Process
represents
represents
represents
1
1
1
Node Group
4720
Node
4721
Service Process
4723

Fig 47

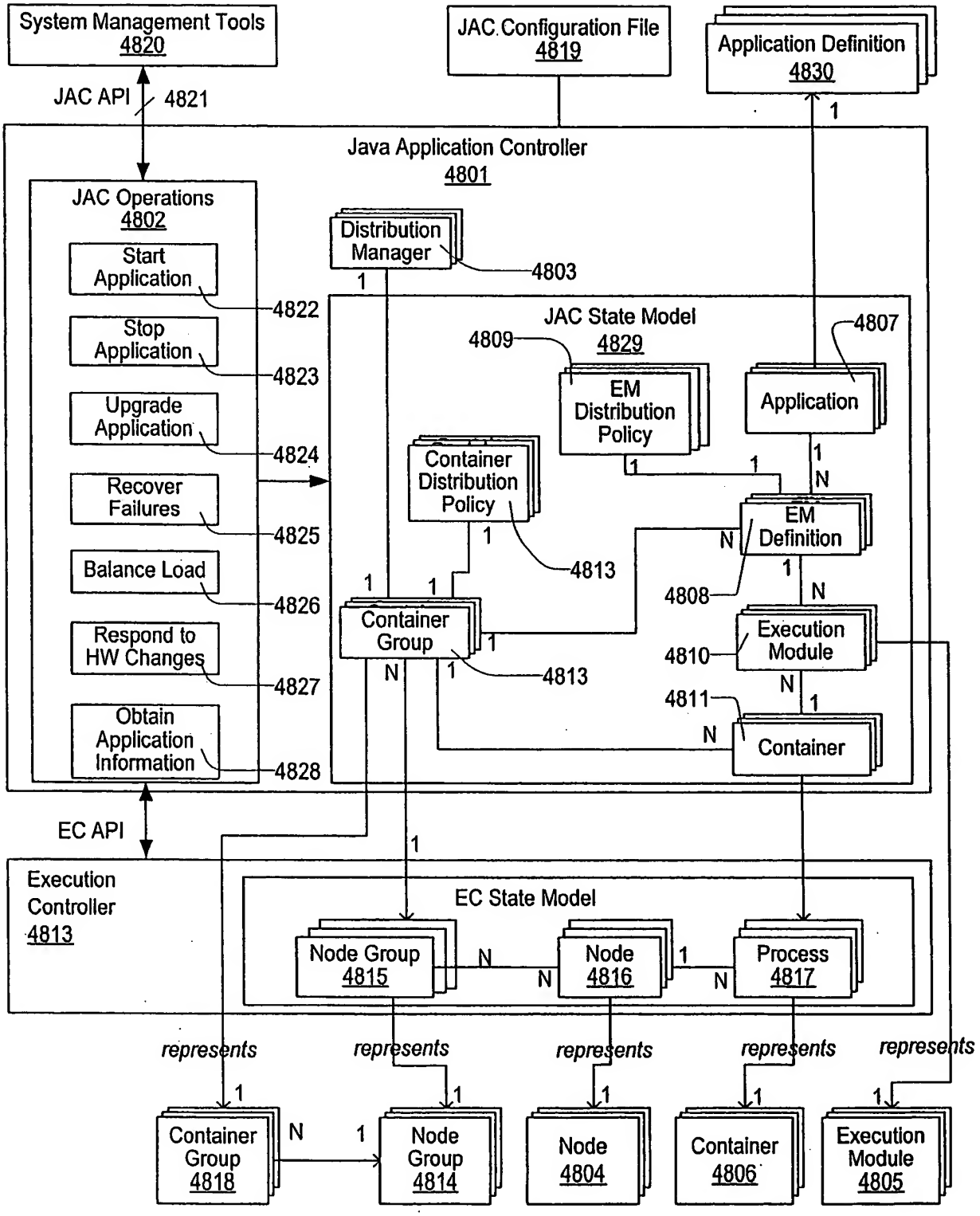

System Management Tools
4820
JAC Configuration File
4819
Application Definition
4830
JAC API
4821
Java Application Controller
4801
JAC Operations
4802
Start Application
4822
Stop Application
4823
Upgrade Application
4824
Recover Failures
4825
Balance Load
4826
Respond to HW Changes
4827
Obtain Application Information
4828
Distribution Manager
4803
JAC State Model
4829
4809
EM Distribution Policy
4807
Application
Container Distribution Policy
4813
EM Definition
4808
Container Group
4813
Execution Module
4810
4811
Container
EC API
Execution Controller
4813
EC State Model
Node Group
4815
Node
4816
Process
4817
represents
represents
represents
represents
represents
Container Group
4818
Node Group
4814
Node
4804
Container
4806
Execution Module
4805

Fig 48